

# 34th Symposium on Theoretical Aspects of Computer Science

STACS 2017, March 8–11, 2017, Hannover, Germany

Edited by

Heribert Vollmer

Brigitte Vallée



*Editors*

Heribert Vollmer	Brigitte Vallée
Institut für Theoretische Informatik	GREYC, UMR CNRS 6072
Leibniz Universität Hannover	Université de Caen Normandie
Hannover, Germany	Caen, France
vollmer@thi.uni-hannover.de	Brigitte.Vallee@unicaen.fr

*ACM Classification 1998*

F.1.1 Models of Computation, F.2.2 Nonnumerical Algorithms and Problems, F.4.1 Mathematical Logic, F.4.3 Formal Languages, G.2.1 Combinatorics, G.2.2 Graph Theory

**ISBN 978-3-95977-028-6**

*Published online and open access by*

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at <http://www.dagstuhl.de/dagpub/978-3-95977-028-6>.

*Publication date*

March, 2017

*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 <http://dnb.d-nb.de>.

*License*

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



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

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

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.STACS.2017.0

ISBN 978-3-95977-028-6

ISSN 1868-8969

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

## LIPICs – Leibniz International Proceedings in Informatics

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

### *Editorial Board*

- Susanne Albers (TU München)
- Chris Hankin (Imperial College London)
- Deepak Kapur (University of New Mexico)
- Michael Mitzenmacher (Harvard University)
- Madhavan Mukund (Chennai Mathematical Institute)
- Catuscia Palamidessi (INRIA)
- Wolfgang Thomas (*Chair*, RWTH Aachen)
- Pascal Weil (CNRS and University Bordeaux)
- Reinhard Wilhelm (Saarland University)

**ISSN 1868-8969**

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



# ■ Contents

Foreword	
<i>Heribert Vollmer and Brigitte Vallée</i> .....	0:ix

## Tutorial

Computational Aspects of Logics in Team Semantics	
<i>Juha Kontinen</i> .....	1:1–1:1

## Invited Talks

Recompression: New Approach to Word Equations and Context Unification	
<i>Artur Jeż</i> .....	2:1–2:3
Discrete Logarithms in Small Characteristic Finite Fields: a Survey of Recent Advances	
<i>Antoine Joux</i> .....	3:1–3:1
Applications of Algorithmic Metatheorems to Space Complexity and Parallelism	
<i>Till Tantau</i> .....	4:1–4:4

## Regular Contributions

Split Contraction: The Untold Story	
<i>Akanksha Agrawal, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi</i> .....	5:1–5:14
The Operator Approach to Entropy Games	
<i>Marianne Akian, Stéphane Gaubert, Julien Grand-Clément, and Jérémie Guillaud</i> .....	6:1–6:14
Parameterized Complexity of Small Weight Automorphisms	
<i>Vikraman Arvind, Johannes Köbler, Sebastian Kuhnert, and Jacobo Torán</i> .....	7:1–7:13
What Can Be Verified Locally?	
<i>Alkida Balliu, Gianlorenzo D’Angelo, Pierre Fraigniaud, and Dennis Olivetti</i> .....	8:1–8:13
Improved Time-Space Trade-Offs for Computing Voronoi Diagrams	
<i>Bahareh Banyassady, Matias Korman, Wolfgang Mulzer, André van Renssen, Marcel Roeloffzen, Paul Seiferth, and Yannik Stein</i> .....	9:1–9:14
Energy-Efficient Delivery by Heterogeneous Mobile Agents	
<i>Andreas Bärtschi, Jérémie Chalopin, Shantanu Das, Yann Disser, Daniel Graf, Jan Hackfeld, and Paolo Penna</i> .....	10:1–10:14
Towards Tighter Space Bounds for Counting Triangles and Other Substructures in Graph Streams	
<i>Suman K. Bera and Amit Chakrabarti</i> .....	11:1–11:14
On Polynomial Approximations Over $\mathbb{Z}/2^k\mathbb{Z}$	
<i>Abhishek Bhrushundi, Prahladh Harsha, and Srikanth Srinivasan</i> .....	12:1–12:12



$\exists\mathbb{R}$ -Complete Decision Problems about Symmetric Nash Equilibria in Symmetric Multi-Player Games <i>Vittorio Bilò and Marios Mavronicolas</i> .....	13:1–13:14
On Büchi One-Counter Automata <i>Stanislav Böhm, Stefan Göller, Simon Halfon, and Piotr Hofman</i> .....	14:1–14:13
Optimizing Tree Decompositions in MSO <i>Mikołaj Bojańczyk and Michał Pilipczuk</i> .....	15:1–15:13
Complexity of Token Swapping and its Variants <i>Édouard Bonnet, Tillmann Miltzow, and Paweł Rzążewski</i> .....	16:1–16:14
Monte Carlo Computability <i>Vasco Brattka, Rupert Hölzl, and Rutger Kuyper</i> .....	17:1–17:14
The Parameterized Complexity of Finding a 2-Sphere in a Simplicial Complex <i>Benjamin Burton, Sergio Cabello, Stefan Kratsch, and William Pettersson</i> .....	18:1–18:14
On Long Words Avoiding Zimin Patterns <i>Arnaud Carayol and Stefan Göller</i> .....	19:1–19:13
Extended Learning Graphs for Triangle Finding <i>Titouan Carette, Mathieu Laurière, and Frédéric Magniez</i> .....	20:1–20:14
Lower Bounds for Elimination via Weak Regularity <i>Arkadev Chattopadhyay, Pavel Dvořák, Michal Koucký, Bruno Loff, and Sagnik Mukhopadhyay</i> .....	21:1–21:14
Parameterized and Approximation Results for Scheduling with a Low Rank Processing Time Matrix <i>Lin Chen, Dániel Marx, Deshi Ye, and Guochuan Zhang</i> .....	22:1–22:14
Fractional Coverings, Greedy Coverings, and Rectifier Networks <i>Dmitry Chistikov, Szabolcs Iván, Anna Lubiw, and Jeffrey Shallit</i> .....	23:1–23:14
Separability of Reachability Sets of Vector Addition Systems <i>Lorenzo Clemente, Wojciech Czerwiński, Sławomir Lasota, and Charles Paperman</i> .....	24:1–24:14
Counting Edge-Injective Homomorphisms and Matchings on Restricted Graph Classes <i>Radu Curticapean, Holger Dell, and Marc Roth</i> .....	25:1–25:15
Robust and Adaptive Search <i>Yann Disser and Stefan Kratsch</i> .....	26:1–26:14
Graphic TSP in Cubic Graphs <i>Zdeněk Dvořák, Daniel Král', and Bojan Mohar</i> .....	27:1–27:13
Independent Sets near the Lower Bound in Bounded Degree Graphs <i>Zdeněk Dvořák and Bernard Lidický</i> .....	28:1–28:13
Semialgebraic Invariant Synthesis for the Kannan-Lipton Orbit Problem <i>Nathanaël Fijalkow, Pierre Ohlmann, Joël Ouaknine, Amaury Pouly, and James Worrell</i> .....	29:1–29:13

The First-Order Logic of Hyperproperties <i>Bernd Finkbeiner and Martin Zimmermann</i> .....	30:1–30:14
Improving and Extending the Testing of Distributions for Shape-Restricted Properties <i>Eldar Fischer, Oded Lachish, and Yadu Vasudev</i> .....	31:1–31:14
Matrix Rigidity from the Viewpoint of Parameterized Complexity <i>Fedor V. Fomin, Daniel Lokshantov, S. M. Meesum, Saket Saurabh, and Meirav Zehavi</i> .....	32:1–32:14
Deterministic Regular Expressions with Back-References <i>Dominik D. Freydenberger and Markus L. Schmid</i> .....	33:1–33:14
On the Decomposition of Finite-Valued Streaming String Transducers <i>Paul Gallot, Anca Muscholl, Gabriele Puppis, and Sylvain Salvati</i> .....	34:1–34:14
Circuit Evaluation for Finite Semirings <i>Moses Ganardi, Danny Hucke, Daniel König, and Markus Lohrey</i> .....	35:1–35:14
Combining Treewidth and Backdoors for CSP <i>Robert Ganian, M. S. Ramanujan, and Stefan Szeider</i> .....	36:1–36:17
On the Complexity of Partial Derivatives <i>Ignacio Garcia-Marco, Pascal Koiran, Timothée Pecatte, and Stéphan Thomassé</i> .	37:1–37:13
Set Membership with Non-Adaptive Bit Probes <i>Mohit Garg and Jaikumar Radhakrishnan</i> .....	38:1–38:13
Pro-Aperiodic Monoids via Saturated Models <i>Samuel J. v. Gool and Benjamin Steinberg</i> .....	39:1–39:14
Trimming and Gluing Gray Codes <i>Petr Gregor and Torsten Mütze</i> .....	40:1–40:14
Mixing of Permutations by Biased Transposition <i>Shahrad Haddadan and Peter Winkler</i> .....	41:1–41:13
Efficient Quantum Walk on the Grid with Multiple Marked Elements <i>Peter Høyer and Mojtaba Komeili</i> .....	42:1–42:14
On OBDD-Based Algorithms and Proof Systems That Dynamically Change Order of Variables <i>Dmitry Itsykson, Alexander Knop, Andrey Romashchenko, and Dmitry Sokolov</i> ...	43:1–43:14
Multiple Random Walks on Paths and Grids <i>Andrej Ivašković, Adrian Kosowski, Dominik Pajak, and Thomas Sauerwald</i> .....	44:1–44:14
On the Size of Lempel-Ziv and Lyndon Factorizations <i>Juha Kärkkäinen, Dominik Kempa, Yuto Nakashima, Simon J. Puglisi, and Arseny M. Shur</i> .....	45:1–45:13
Voting and Bribing in Single-Exponential Time <i>Dušan Knop, Martin Kouřtecký, and Matthias Mnich</i> .....	46:1–46:14
A Complexity Dichotomy for Poset Constraint Satisfaction <i>Michael Kompatscher and Trung Van Pham</i> .....	47:1–47:12

Structural Properties and Constant Factor-Approximation of Strong Distance- $r$ Dominating Sets in Sparse Directed Graphs <i>Stephan Kreuzer, Roman Rabinovich, Sebastian Siebertz, and Grischa Weberstädt</i> .....	48:1–48:15
Computing Majority by Constant Depth Majority Circuits with Low Fan-in Gates <i>Alexander S. Kulikov and Vladimir V. Podolskii</i> .....	49:1–49:14
Minkowski Games <i>Stéphane Le Roux, Arno Pauly, and Jean-François Raskin</i> .....	50:1–50:13
On the Sensitivity Complexity of $k$ -Uniform Hypergraph Properties <i>Qian Li and Xiaoming Sun</i> .....	51:1–51:12
The Complexity of Knapsack in Graph Groups <i>Markus Lohrey and Georg Zetsche</i> .....	52:1–52:14
Algorithmic Information, Plane Kakeya Sets, and Conditional Dimension <i>Jack H. Lutz and Neil Lutz</i> .....	53:1–53:13
On the Synchronisation Problem over Cellular Automata <i>Gaétan Richard</i> .....	54:1–54:13
Word Equations Where a Power Equals a Product of Powers <i>Aleksi Saarela</i> .....	55:1–55:9
Improved Distance Queries and Cycle Counting by Frobenius Normal Form <i>Piotr Sankowski and Karol Węgrzycki</i> .....	56:1–56:14
Lower Bounds on Key Derivation for Square-Friendly Applications <i>Maciej Skorski</i> .....	57:1–57:12
List Approximation for Increasing Kolmogorov Complexity <i>Marius Zimand</i> .....	58:1–58:12



## ■ Foreword

The Symposium on Theoretical Aspects of Computer Science conference series is an international forum for original research on theoretical computer science. Typical areas are:

- algorithms and data structures, including: design of parallel, distributed, approximation, 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 and verification, rewriting and deduction;
- current challenges, for example: natural computing, quantum computing, mobile and net computing.

STACS is held alternately in France and in Germany. This year's conference (taking place March 8–11 in Hannover) is the 34th 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), and Orléans (2016).

The interest in STACS has remained at a high level over the past years. The STACS 2017 call for papers led to 212 submissions with authors from 38 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. For the third 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 the many external referees for their valuable work. The overall very high quality of the submissions made the selection a difficult task, and there were intense and interesting discussions inside the program committee.

This year, the conference includes a tutorial. We would like to express our thanks to the speaker Juha Kontinen for this tutorial, as well as to the invited speakers, Artur Jež, Antoine Joux, and Till Tantau. Special thanks go to the local organizing committee for continuous help throughout the conference organization.

Moreover, we thank Marc Herbstritt 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 tutorial. 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 2017 has received funds and help from the Deutsche Forschungsgemeinschaft (DFG), for which we are very grateful.

Hannover and Caen, March 2017

Heribert Vollmer and Brigitte Vallée

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).  
Editors: Heribert Vollmer and Brigitte Vallée



Leibniz International Proceedings in Informatics

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



SYMPOSIUM  
ON THEORETICAL  
ASPECTS  
OF COMPUTER  
SCIENCE



## ■ Conference Organization

### Program Committee

Olaf Beyedrsdorff	University of Leeds
Francine Blanchet-Sadri	University of North Carolina
Beate Bollig	Technische Universität Dortmund
Nicolas Bonichon	Université Bordeaux 1
Jean Cardinal	Université Libre de Bruxelles
Nicolò Cesa-Bianchi	Università degli Studi di Milano
Léo Ducas	Centrum Wiskunde en Informatica
Arnaud Durand	Université Denis Diderot - Paris 7
Michael Elberfeld	RWTH Aachen
Pinar Heggernes	Universitetet i Bergen
Martin Hofer	Goethe-Universität Frankfurt am Main
Giuseppe Italiano	Università degli Studi di Roma "Tor Vergata"
Emmanuel Jeandel	Université de Lorraine
Christian Komusiewicz	Friedrich-Schiller-Universität Jena
Nutan Limaye	Indian Institute of Technology, Bombay
Florin Manea	Christian-Albrechts-Universität zu Kiel
Filip Murlak	Uniwersytet Warszawski
Seth Pettie	University of Michigan
Irena Rusu	Université de Nantes
Sylvain Schmitz	Université Paris-Saclay
Frank Stephan	National University of Singapore
Brigitte Vallée	Université de Caen (co-chair)
Heribert Vollmer	Leibniz Universität Hannover (co-chair)
Ronald de Wolf	Centrum Wiskunde en Informatica
Florian Zuleger	Technische Universität Wien



**Local Organization Committee**

Maurice Chandoo

Anselm Haak

Martin Lück

Arne Meier (chair)

Anca Vais

Heribert Vollmer

## External Reviewers

Amir Abboud	Marco Bressan	Jing Deng
Fariad Abu Zaid	Dan Browne	William E. Devanny
Ali Akhavi	Véronique Bruyère	Olivier Devillers
Eric Allender	Niv Buchbinder	Ajit Diwan
Joshua Alman	Kevin Buchin	Amina Doumane
Andris Ambainis	Binh-Minh Bui-Xuan	Stefan Droste
Benjamin Aminof	Laurent Bulteau	Philippe Duchon
Alex Andoni	Jaroslav Byrka	Christoph Dürr
Patrizio Angelini	Pavel Čadek	Matteo Dusefante
Srinivasan Arunachalam	Michaël Cadilhac	Zdenek Dvorak
Vikraman Arvind	Cristian S. Calude	Martin Dyer
Eugene Asarin	Florent Capelli	Marcin Dziubiński
Yossi Azar	Katarina Cechlarova	Charilaos Efthymiou
Nicolas Bacquey	Sourav Chakraborty	Thorsten Ehlers
Nikhil Balaji	Maurice Chandoo	Kord Eickmeyer
Eric Balkanski	Chandra Chekuri	Khaled Elbassioni
Grey Ballard	Jiehua Chen	Tapio Elomaa
Sayan Bandyopadhyay	Ruiwen Chen	Leah Epstein
Aritra Banik	Alexey Chernov	Marco Faella
Leonid Barenboim	Ho Yee Cheung	Piotr Faliszewski
David Mix Barrington	Dmitry Chistikov	Angelo Fanelli
Nicolas Basset	Rajesh Chitnis	Lene Favrholdt
Cristina Bazgan	Ferdinando Cicalese	Serge Fehr
Michael Bekos	Lorenzo Clemente	Stefan Felsner
Rémy Belmonte	Julien Clément	Henning Fernau
Aleksandrs Belovs	David Cohen	Guillaume Fertin
Huxley Bennett	Michael Cohen	Hendrik Fichtenberger
Cédric Bentz	Michelle Cordier	Nathanaël Fijalkow
Christoph Berkholz	Pierluigi Crescenzi	Till Fluschnik
Dietmar Berwanger	Christophe Crespelle	Fedor Fomin
René Van Bevern	Ágnes Cseh	Hervé Fournier
Bhaswar Bhattacharya	James Currie	Nathanaël François
Abhishek Bhowmick	Marek Cygan	Dominik D. Freydenberger
Stella Biderman	Wojciech Czerwiński	Takuro Fukunaga
Marcin Bienkowski	Zhu Daming	Radoslav Fulek
Ahmad Biniiaz	Luc Dartois	Hortensia Galeana-Sanchez
Markus Bläser	Bireswar Das	Pierre Ganty
Dakota Blair	Samir Datta	Jugal Garg
Joshua Blinkhorn	Laure Daviaud	Leszek Gasieniec
Hans L. Bodlaender	Claire David	Serge Gaspers
Martin Böhm	Adam Day	Pawel Gawrychowski
Benedikt Bollig	Joel Day	Sutanu Gayen
Ilario Bonacina	Jean-Lou De Carufel	Loukas Georgiadis
Ralph Bottesche	Mateus De Oliveira Oliveira	Dan Ghica
Joan Boyar	Martin Delacourt	Archontia Giannopoulou
Andreas Brandstädt	Argyrios Deligkas	Vasilis Gkatzelis
Vladimir Braverman	Holger Dell	Amy Glen

Marc Glisse	Juhani Karhumaki	Florent Madelaine
Tomasz Gogacz	Jarkko Kari	Frederic Maffray
Kira Goldner	Elham Kashefi	Frederic Magniez
Isaac B. Goldstein	Jens Katelaan	Meena Mahajan
Petr Golovach	Telikepalli Kavitha	Andreas Maletti
Themistoklis Gouleakis	Neeraj Kayal	Sebastian Maneth
Fabrizio Grandoni	Thomas Kesselheim	David Manlove
Noam Greenberg	Sandra Kiefer	Weizhen Mao
Sander Gribling	Yusuke Kobayashi	Irène Marcovici
Radu Grigore	Bojana Kodric	Jean-Yves Marion
Serge Grigorieff	Pascal Koiran	Nicolas Markey
Martin Groß	Pavel Kolev	Euripides Markou
Jiong Guo	Igor Konnov	Barnaby Martin
Anselm Haak	Juha Kontinen	Dániel Marx
Peter Habermehl	Wouter M. Koolen	Claire Mathieu
Michel Habib	Eryk Kopczynski	Elvira Mayordomo
Serge Haddad	Guy Kortsarz	Ernst W. Mayr
Simon Halfon	Dmitry Kosolobov	Andrew McGregor
Michael Hanus	Michal Koucky	Pierre Mckenzie
Sariel Har-Peled	Marcin Kozik	Ian McQuillan
Prahladh Harsha	Jan Kratochvil	Moti Medina
Hamed Hatami	Stefan Kratsch	Kitty Meeks
Meng He	Matthias Krause	Klaus Meer
Lauri Hella	Sebastian Krinninger	Arne Meier
Miki Hermann	Amer Krivosija	Or Meir
Danny Hermelin	Alexander Kulikov	Stefan Mengel
Claudio Hermida	Raghav Kulkarni	Robert Mercas
Luke Hinde	Neeraj Kumar	George Mertzios
Udo Hoffmann	Nirman Kumar	David Mezlaf
Mathieu Hoyrup	Anthony Labarre	Mehdi Mhalla
Daniel Hsu	Arnaud Labourel	Pierre Michaud
Chien-Chung Huang	Guillaume Lagarde	Tillmann Miltzow
Danny Hucke	Victor Lagerqvist	Neeldhara Misra
Hsien-Kuei Hwang	Harry Lang	Victor Mitrană
Rasmus Ibsen-Jensen	Jérôme Lang	Rajat Mittal
David Ilcinkas	Sophie Laplante	Shuichi Miyazaki
Kazuo Iwama	Silvio Lattanzi	Matthias Mnich
Rahul Jain	Massimo Lauria	Tobias Mömke
Damien Jamet	Marijana Lazic	Hendrik Molter
Bart M. P. Jansen	Francois Le Gall	Raffaele Mosca
Klaus Jansen	Stephane Le Roux	Amer Mouawad
Jesper Jansson	Mathieu Liedloff	Haiko Müller
Maximilian Jaroschek	Zhixin Liu	Partha Mukhopadhyay
Mark Jerrum	Christof Löding	Ian Munro
Peter Jonsson	Andrew Lohr	Alexander Munteanu
Hossein Jowhari	Markus Lohrey	Andrzej Murawski
Stasys Jukna	Veronika Loitzenbauer	Pareesh Nakhe
Naonori Kakimura	Satyanarayana Lokam	Meghana Nasre
Lior Kamma	Daniel Lokshtanov	Jesper Nederlof
Mamadou Moustapha Kanté	Zvi Lotker	Daniel Neider
Michael Kapralov	Martin Lück	Cyril Nicaud
Frantisek Kardos	Robert Lukotka	André Nichterlein

Andre Nies	Wojciech Rytter	Jim Tao
Nicolas Nisse	Ville Salo	Sébastien Tavenas
Dirk Nowotka	Fernando Sanchez Villaamil	Raghunath Tewari
Jerri Nummenpalo	Arnaud Sangnier	Johan Thapper
Sebastian Ordyniak	Swagato Sanyal	Guillaume Theyssier
Sigal Oren	Jayalal Sarma	Thomas Thierauf
Patrice Ossona de Mendez	Srinivasa Rao Satti	Sumedh Tirodkar
Maris Ozols	Martin Sauerhoff	Luca Trevisan
Rasmus Pagh	Nitin Saurabh	Madhur Tulsiani
Dominik Pajak	Saket Saurabh	Jerzy Tyszkiewicz
Greta Pangborn	Guillaume Scerri	Alexander Ushakov
Thomas Pani	John Schanck	Przemysław Uznański
Fahad Panolan	Bruno Scherrer	Vincent Vajnovszki
Charis Papadopoulos	Kevin Schewior	Pierre Valarcher
Charles Paperman	Markus L. Schmid	Joran van Apeldoorn
Nikos Parotsidis	Henning Schnoor	Joris van der Hoeven
Aline Parreau	Uwe Schöning	Erik Jan van Leeuwen
Ludovic Patey	Pascal Schweitzer	André van Renssen
Christophe Paul	Chris Schwiegelshohn	Rob van Stee
Daniel Paulusma	Stefan Schwoon	Anke van Zuylen
Arno Pauly	Olivier Serre	Vinodchandran Variyam
A Pavan	Hadas Shachnai	Sergei Vassilvitskii
Romain Pechoux	Rudrapatna Shyamasundar	Yadu Vasudev
Pan Peng	Aaron Sidford	Daniel Vaz
Richard Peng	Sebastian Siebertz	Sander Verdonschot
Sylvain Perifel	Rodrigo Silveira	Nikolay Vereshchagin
Will Perkins	Sean Simmons	Sergey Verlan
Daniela Petrisan	Moritz Sinn	Stéphane Vialette
Ramchandra Phawade	Johan Sivertsen	Ben Lee Volk
Geevarghese Philip	Alexander Skopalik	Pooja Vyavahare
Marcin Pilipczuk	Michał Skrzypczak	Jan Philipp Wächter
Michał Pilipczuk	Michiel Smid	Magnus Wahlström
Sophie Pinchinat	Kian Wee Soh	Michael Walter
Thomas Place	Manuel Sorge	Andreas Weiermann
Vladimir Podolskii	Joachim Spoerhase	Oren Weimann
Danny Bøgsted Poulsen	Srikanth Srinivasan	S. Matthew Weinberg
Tobias Pröger	Konstantinos Stavropoulos	Georg Weissenbacher
Kirk Pruhs	Benjamin Steinberg	Matthias Westermann
Ivan Radicek	Brett Stevens	Andreas Winter
Narad Rampersad	Irina Stoilkovska	Carsten Witt
Raghavendra Rao B V	Darren Strash	Maximilian Wötzel
Michael Rao	Howard Straubing	Marcin Wrochna
Ilya Razenshteyn	Yann Strozecki	Christian Wulff-Nilsen
Daniel Reidenbach	Hsin-Hao Su	Junjie Ye
Anja Rey	Ondrej Suchy	Yu Yu
Leonid Reyzin	He Sun	Jean-Baptiste Yunès
Colin Riba	Jukka Suomela	Marc Zeitoun
Dana Richards	Subash Suri	Georg Zetsche
Jérémie Roland	Nimrod Talmon	Thomas Zeume
Andrei Romashchenko	Till Tantau	Chihao Zhang





# Computational Aspects of Logics in Team Semantics

Juha Kontinen

Department of Mathematics and Statistics, University of Helsinki, Helsinki, Finland

juha.kontinen@helsinki.fi

---

## Abstract

Team Semantics is a logical framework for the study of various dependency notions that are important in many areas of science. The starting point of this research is marked by the publication of the monograph *Dependence Logic* (Jouko Väänänen, 2007) in which first-order dependence logic is developed and studied. Since then team semantics has evolved into a flexible framework in which numerous logics have been studied.

Much of the work in team semantics has so far focused on results concerning either axiomatic characterizations or the expressive power and computational aspects of various logics. This tutorial provides an introduction to team semantics with a focus on results regarding expressivity and computational aspects of the most prominent logics of the area. In particular, we discuss dependence, independence and inclusion logics in first-order, propositional, and modal team semantics. We show that first-order dependence and independence logic are equivalent with existential second-order logic and inclusion logic with greatest fixed point logic. In the propositional and modal settings we characterize the expressive power of these logics by so-called team bisimulations and determine the complexity of their model checking and satisfiability problems.

**1998 ACM Subject Classification** F.4.1 Mathematical Logic

**Keywords and phrases** team semantics, dependence logic, model checking, satisfiability problem, team bisimulation

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.1

**Category** Tutorial



© Juha Kontinen;

licensed under Creative Commons License CC-BY

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 1; pp. 1:1–1:1

Leibniz International Proceedings in Informatics



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



# Recompression: New Approach to Word Equations and Context Unification

Artur Jeż

Institute of Computer Science, University of Wrocław, Wrocław, Poland

---

## Abstract

Word equations is given by two strings over disjoint alphabets of letters and variables and we ask whether there is a substitution that satisfies this equation. Recently, a new PSPACE solution to this problem was proposed, it is based on compressing simple substrings of the equation and modifying the equation so that such operations are sound. The analysis focuses on the way the equation is stored and changed rather than on the combinatorics of words. This approach greatly simplified many existing proofs and algorithms. In particular, unlike the previous solutions, it generalises to equations over contexts (known for historical reasons as context unification): contexts are terms with one special symbol that represent a missing argument and they can be applied on terms, in which case their argument replaces the special constant.

**1998 ACM Subject Classification** F.4.3 [Formal Languages] Decision Problems, F.4.2 Grammars and Other Rewriting Systems, F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** Word equations, exponent of periodicity, semantic unification, string unification, context unification, compression

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.2

**Category** Invited Talk

## 1 Recompression for word equations

Word equations is given by two strings  $(U, V)$  over disjoint alphabets of letters  $(\Sigma)$  and variables  $(\mathcal{X})$ . We are to decide, whether there is a substitution  $S : \mathcal{X} \rightarrow \Sigma^*$  which turns  $U$  and  $V$  into equal strings over  $\Sigma$ . This problem was first solved by Makanin [6], both the algorithm and its analysis employs nontrivial facts from word combinatorics. It took 20 years for a different solution to emerge, it was proposed by Plandowski [7]. It is based on a different approach, still uses nontrivial word combinatorics concepts and additionally uses compression; this algorithm works in PSPACE, which is the best computational complexity bound known till today, and the problem is only known to be NP-hard.

The recompression approach [5], which I will survey in this talk, takes a different approach. It focuses mostly on the compression and on the way equation is represented, and not at all on the word combinatorics. On a high level we want to perform two compression operations on the sides of the equation after applying the substitution to variables, which we denote by  $S(U)$  and  $S(V)$ :

- the  $a$ -block compression replaces every maximal block  $a^\ell$  with a (fresh) letter  $a_\ell$  (an  $a$ -block is a subsequence consisting of letters  $a$  alone and that cannot be extended by  $a$  nor to the left, nor to the right);
- the  $ab$  pair compression, where  $a \neq b$ , replaces every occurrence of a pair  $ab$  with a fresh letter  $c$ .

For instance, given a word  $aaabababaabaa$  the  $ab$  pair compression returns  $aaccaacaa$  while  $a$  block compression:  $a_3bababa_3ba_2$ .



© Artur Jeż;  
licensed under Creative Commons License CC-BY

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 2; pp. 2:1–2:3

Leibniz International Proceedings in Informatics



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

In order to apply those operations, we may need to process the equation appropriately: given a solution  $S$  we say that  $ab$  is a *crossing pair* (with respect to  $S$ ) if there is an occurrence of  $ab$  in  $S(U)$  or  $S(V)$  that does not fully come from the equation or from a substitution for one occurrence of a variable. A pair is *noncrossing* (with respect to  $S$ ) otherwise. If  $ab$  is noncrossing with respect to a solution  $S$  of an equation  $U = V$ , then performing  $ab$ -compression on sides of the equation results in an equation  $U' = V'$  that has a corresponding solution  $S'$ , which is obtained by also replacing every  $ab$  in  $S(X)$  by  $c$ . Then  $S'(U')$  and  $S'(V')$  are obtained from  $S(U)$  and  $S(V)$  by  $ab$ -pair compression, as desired.

If  $ab$  is crossing, we *uncross it*: for every variable  $X$  we nondeterministically guess, whether  $S(X)$  begins with  $b$  and if so, we replace  $X$  with  $bX$  and whether  $S(X)$  ends with  $a$ , in which case we replace  $X$  with  $Xa$ . It is easy to show that after doing so for all variables, the pair  $ab$  is noncrossing, and so it can be compressed.

We treat blocks similarly: when performing the  $a$  block compression, we look at maximal blocks of  $a$  in  $S(U)$  and  $S(V)$  that come partially from the equation and partially from substitution for variables. This time, though, adding one letter may be not enough, we replace  $X$  with  $a^\ell X a^r$ , where  $S(X) = a^\ell w a^r$  and  $w$  does not begin, nor end, with  $a$ . Again, performing this for all variables makes it possible to perform the  $a$ -block compression.

It is easy to see that our procedure is sound and complete, what remains is a termination proof, we show it by proving a  $4n^2$  bound on stored equation (for appropriate non-deterministic choices). To this end we give a strategy for choosing a pair/block to compress: If there is a non-crossing pair or  $a$  without crossing blocks, we perform the corresponding compression. If not, then we choose the pair/block after compression of which the equation is the shortest.

Observe that if occurrences of a pair/blocks of a letter use  $\alpha$  letters in the equation, then compression of this pair/blocks of a letter removes at least  $\alpha/2$  letters from the equation. On the other hand, each uncrossing introduces at most 2 letters per variable, so at most  $2n$  in total. As each crossing pair/letter with a crossing block corresponds to a left or right side of an occurrence of a variable in the equation, in total there are at most  $2n$  different crossing pairs and letters with crossing block. Thus if the equation has length  $m$  and it has no noncrossing pairs and no letters without crossing blocks, for some crossing pair/letter with a crossing block the length of the equation after corresponding the compression is at most  $m' = (1 - \frac{1}{2n})m + 2n$ . It is easy to check that if  $m \leq 4n^2$  then also  $m' \leq 4n^2$ .

## 2 Context unification

When one considers terms instead of words, equations become more involved and the first question is: What can the variables represent? If we allow the variables to represent only ground (closed) terms, then we end up with *first order unification*, for which the polynomial time algorithm is folklore. Allowing the variables to represent functions leads to second order unification, which is undecidable [3].

Context unification [1, 2, 8] is a fragment of second-order unification: A *context* is a ground term with exactly one occurrence of a special constant ‘•’ that represents a missing argument; one should think of ‘•’ as a ‘hole’ to be instantiated by a ground term later on: Contexts can be applied to ground terms, which results in a replacement of ‘•’ by the given ground term; similarly we can define a composition of two contexts, which is again a context. In *context unification* we allow the variables to represent contexts; syntactically, the variables are treated as unary function symbols. A solution assigns to each (context) variable a context such that both sides of the equation evaluate to the same (ground) term. For ease of use, we also allow ground variables that represent ground terms, syntactically they are treated as constants and the solution substitutes them with ground terms.

The decidability status of context unification was unknown for years, as on one hand the undecidability proofs for second order unification essentially used the fact that argument can be used unbounded number of times, and on the other hand the combinatorial properties of strings used by algorithms of Makanin and Plandowski had no equivalent in the term case.

The recompression approach and the corresponding algorithm generalise naturally to the term case: the two compression operations can be readily applied to consecutive unary nodes. One still needs additional operation: leaves compression. The  $(f, i, c)$  leaves compression replaces each subterm  $f(t_1, \dots, t_{i-1}, c, t_{i+1}, \dots, t_m)$  with  $f(t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_m)$ . All notions from pairs and blocks generalise to this new compression operation: The crossing  $(f, i, c)$  is defined similarly, though the corresponding uncrossing is more involved: on one hand, if a (ground) variable  $x$  has a solution  $S(x) = c$  and it has an occurrence in which  $x$  is the  $i$ -th child, then we replace  $x$  with  $c$ . On the other hand, when in  $S(X)$  the symbol above ‘•’ is  $f$  and ‘•’ is the  $i$ -th child, we replace  $X$  with  $X(f(x_1, \dots, x_{i-1}, \bullet, x_{i+1}, \dots, x_m))$ , where  $x_1, \dots, x_{i-1}, x_{i+1}, x_m$  are fresh variables.

The algorithm is similar as before: we guess a compression, perform the uncrossing if needed and then compress. With appropriate choices the equation stays polynomial. Some additional analysis is needed to bound the number of introduced fresh variables. Thus the satisfiability of context unification is in PSPACE [4]; the lower bound is NP, as for words’ case.

---

## References

- 1 Hubert Comon. Completion of rewrite systems with membership constraints. Part I: Deduction rules. *J. Symb. Comput.*, 25(4):397–419, 1998. doi:10.1006/jsc.1997.0185.
- 2 Hubert Comon. Completion of rewrite systems with membership constraints. Part II: Constraint solving. *J. Symb. Comput.*, 25(4):421–453, 1998. doi:10.1006/jsc.1997.0186.
- 3 Warren D. Goldfarb. The undecidability of the second-order unification problem. *Theor. Comput. Sci.*, 13:225–230, 1981. doi:10.1016/0304-3975(81)90040-2.
- 4 Artur Jež. Context unification is in PSPACE. In Elias Koutsoupias, Javier Esparza, and Pierre Fraigniaud, editors, *ICALP*, volume 8573 of *LNCS*, pages 244–255. Springer, 2014. full version at <http://arxiv.org/abs/1310.4367>. doi:10.1007/978-3-662-43951-7\_21.
- 5 Artur Jež. Recompression: a simple and powerful technique for word equations. *Journal of the ACM*, 63(1):4:1–4:51, Feb 2016. doi:10.1145/2743014.
- 6 Gennadii Makanin. The problem of solvability of equations in a free semigroup. *Matematicheskii Sbornik*, 2(103):147–236, 1977. (in Russian).
- 7 Wojciech Plandowski. Satisfiability of word equations with constants is in PSPACE. *J. ACM*, 51(3):483–496, 2004. doi:10.1145/990308.990312.
- 8 Manfred Schmidt-Schauß. Unification of stratified second-order terms. Internal Report 12/94, Johann-Wolfgang-Goethe-Universität, 1994.



# Discrete Logarithms in Small Characteristic Finite Fields: a Survey of Recent Advances\*

Antoine Joux

Sorbonne Universités, UPMC Université Paris 6, Paris, France  
Antoine.Joux@m4x.org

---

## Abstract

The discrete logarithm problem is one of the few hard problems on which public-key cryptography can be based. It was introduced in the field by the famous Diffie–Hellman key exchange protocol. Initially, the cryptographic use of the problem was considered in prime fields, but was readily generalized to arbitrary finite fields and, later, to elliptic or higher genus curves.

In this talk, we survey the key technical ideas that can be used to compute discrete logarithms, especially in the case of small characteristic finite fields. These ideas stem from about 40 years of research on the topic. They appeared along the long road that leads from the initial belief that this problem was hard enough for cryptographic purpose to the current state of the art where it can no longer be considered for cryptographic use. Indeed, after the recent developments started in 2012, we now have some very efficient practical algorithms to solve this problem. Unfortunately, these algorithms remain heuristic and one important direction for future research is to lift the remaining heuristic assumptions.

**1998 ACM Subject Classification** F.2.1 Numerical Algorithms and Problems

**Keywords and phrases** Cryptography, Discrete logarithms, Finite fields

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.3

**Category** Invited Talk

---

\* This work has been supported in part by the European Union's H2020 Programme under grant agreement number ERC-669891.



© Antoine Joux;

licensed under Creative Commons License CC-BY

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 3; pp. 3:1–3:1

Leibniz International Proceedings in Informatics



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





# Applications of Algorithmic Metatheorems to Space Complexity and Parallelism

Till Tantau

Institute for Theoretical Computer Science, Universität zu Lübeck, Germany  
tantau@tcs.uni-luebeck.de

---

## Abstract

Algorithmic metatheorems state that if a problem can be described in a certain *logic* and the inputs are *structured* in a certain way, then the problem can be solved with a certain *amount of resources*. As an example, by Courcelle’s Theorem all monadic second-order (“in a certain logic”) properties of graphs of bounded tree width (“structured in a certain way”) can be solved in linear time (“with a certain amount of resources”). Such theorems have become a valuable tool in algorithmics: If a problem happens to have the right structure and can be described in the right logic, they immediately yield a (typically tight) upper bound on the time complexity of the problem. Perhaps even more importantly, several complex algorithms rely on algorithmic metatheorems internally to solve subproblems, which considerably broadens the range of applications of these theorems. The talk is intended as a gentle introduction to the ideas behind algorithmic metatheorems, especially behind some recent results concerning space classes and parallel computation, and tries to give a flavor of the range of their applications.

**1998 ACM Subject Classification** F.4.1 Mathematical Logic, G.2.2 Graph Theory

**Keywords and phrases** Algorithmic metatheorems, Courcelle’s Theorem, tree width, monadic second-order logic, logarithmic space, parallel computations

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.4

**Category** Invited Talk

## 1 Talk Summary<sup>1</sup>

Alice, a first-year student of computer science, has an evil homework assignment: Devise an efficient algorithm for the vertex cover problem (at least I feel that tasking first-year students with solving NP-complete problems is a trifle unfair). I guess *you* are right now mentally weighing the different tools at your disposal from the vast machinery developed in complexity theory for attacking such problems – but what would a first-year student do? Being smart, Alice tries to apply the arguably most important and ubiquitous algorithmic approach in computer science: divide-and-conquer. After all, the approach lies at the heart of the fundamental algorithms she just learned about, including merge sort, quick sort, and binary search.

**Solving Vertex Cover Using Divide-And-Conquer?** Naturally, Alice soon notices that the divide-and-conquer approach fails quite miserably when applied to finding small vertex covers.

---

<sup>1</sup> This summary contains some revised material from the introduction of the paper [9]. That paper is a good starting point for a more detailed introduction to algorithmic metatheorems for logarithmic space and circuit classes.



© Till Tantau;

licensed under Creative Commons License CC-BY

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 4; pp. 4:1–4:4

Leibniz International Proceedings in Informatics



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

The problem lies in the *dividing phase*: She finds no way of dividing, for instance, a clique into parts. Alice learns an important lesson here: divide-and-conquer is only applicable to problems whose inputs are “amenable” to dividing them into parts. After telling her professor about her difficulties, she relents and makes the problem (much) easier by restricting the input to *trees*. Now, clearly, dividing the input is no longer a problem: For a tree  $T$  with root  $r$  we can recurse on the subtrees  $T_1$  to  $T_m$  rooted at the children  $c_1$  to  $c_m$  of the root.

Alice still has to tackle the *merging phase* in her divide-and-conquer algorithm: How does one assemble optimal vertex covers for the  $T_i$  into a vertex cover for the whole tree  $T$ ? Clearly, this is not a trivial task since optimal vertex covers for each subtree do not suffice to build an optimal overall vertex cover. The trick is to compute *two* optimal vertex covers for each subtree: one for the case that the root is part of the vertex cover and one for the case that it is not. This yields a divide-and-conquer algorithm for the vertex cover problem on trees that runs in linear time.

**The Question of Why.** Algorithmic metatheorems, which this talk is about, help us in understanding *why* the vertex cover problem behaves the way it does with respect to the divide-and-conquer approach. Why does the division phase fail? Why does the merging phase work? The first question has a fairly easy answer: arbitrary graphs do not have any “decomposition property” at all. On the other hand, trees certainly *can* be decomposed very well, and it turns out that this is still the case when the graph is “nearly” a tree, namely a graph of bounded tree width. The second question seems harder: The answer “because solutions can be assembled using a trick” does not generalize very well. It took the research community quite some time to find a better answer: In 1990, Bruno Courcelle found that the merging phase works “because the vertex cover problem can be described in monadic second-order logic.”

The general pattern underlying algorithmic metatheorems is as follows: If a problem can be described in a certain logic (“are amenable to merging” for the right logic) and instances can be decomposed in a certain tree-wise fashion (“are amenable to division”), they can be solved within a certain amount of time. The first and most famous of these theorems is the just-mentioned Theorem of Courcelle [1]: *All monadic second-order properties of graphs of bounded tree width can be decided in linear time.* A long line of further theorems have later been obtained by varying the three “parameters” of algorithmic metatheorems: the logic, the instance structure, and the required resources. By weakening one of them, one can often strengthen another. For instance, for problems describable in *first-order* logic, we can change the requirement on the decomposition property to, for instance, nowhere dense graphs (a much larger class of graphs than those of bounded tree width) and still obtain a (near) linear time bound [7] or to planar graphs and still obtain a linear time bound [4]. In another direction, when we increase the time bound to polynomial (rather than linear) time, we can broaden the class of graphs to graphs of bounded clique-width [2]. In yet another direction, which this talk will mainly be about, instead of the classical sequential worst-case time bounds, one can look at the space complexity or the parallel complexity. One important result in that direction [3] is that Courcelle’s Theorem also holds when “linear time” is replaced by “logarithmic space.”

**The Range of Applications of Algorithmic Metatheorems.** The power of algorithmic metatheorems lies in their ease of application. Had Alice known about Courcelle’s Theorem, she could have finished her homework much more quickly: The vertex cover problem can be described in monadic second-order logic and trees are clearly very “tree-like,” so the theorem

tells her (and us) that there is a linear-time algorithm for the problem. Admittedly, devising a linear time algorithm for the vertex cover problem on trees is not all that hard – but by the logspace version of Courcelle’s Theorem, we also get a logspace algorithm for this problem for free. You are cordially invited to try to come up with such an algorithm directly (or, failing that, make it a homework assignment for your students).

To make the vertex cover problem accessible to algorithmic metatheorems, we had to insist that all input graphs are trees (a ridiculously strong restriction) or, at least, that they are tree“-like” (no longer a ridiculous restriction, but still a strong restriction). It is thus somewhat surprising that algorithmic metatheorems *can* be used in contexts where the inputs are *not* tree-like graphs. The underlying algorithmic approach is quite ingenious: On input of a graph, if the graph is tree-like, apply an algorithmic metatheorem; and if the graph is not tree-like, it must be “highly cyclic internally,” which we may be able to use to our advantage in solving the problem.

One deceptively simple problem where the just-mentioned approach works particularly well is the *even cycle problem*, which just asks whether there is a cycle of even length in a graph. It is not difficult to show that, like the vertex cover problem and just about any other interesting problem, the even cycle problem can be described in monadic second-order logic. Thus, it can be solved efficiently on tree-like graphs. Now, what about those “highly cyclic” graphs that are *not* tree-like? Intuitively, these many internal cycles might very well make it easier to decide whether the graph has an even cycle. Indeed, they make it *very* easy: such graphs *always* have an even cycle [10]. In other words, we can solve the even cycle problem on *arbitrary* graphs as follows: If the input graph is not tree-like, simply answer “yes,” otherwise apply Courcelle’s Theorem to it.

We will not always be so lucky that the to-be-solved problem more or less disappears for non-tree-like graphs, but in the talk several interesting problems are presented where algorithmic metatheorem play a key role in the internals of algorithms.

**Related Work.** As the title suggests, this talk focuses on algorithmic metatheorems for *space classes* and *parallel* computation (mainly in the form of circuits of polylogarithmic depth). In contrast, most algorithmic metatheorems in the literature concern *time classes*. There are a number of excellent surveys on algorithmic metatheorems and their applications regarding time-efficient computations [5, 6, 8]. An interesting aspect of the theorems covered in the talk is that they can be used to establish *completeness results* for many problems for which the classical algorithmic metatheorems do not yield an exact complexity-theoretic classification: Using Courcelle’s Theorem and the tricks hinted at earlier, the even cycle problem can be solved in linear time and, clearly, this is also a tight lower bound. However, from a structural complexity-theoretic point of view, the problem is most likely not *complete* for linear time; indeed, it *is* complete for logarithmic space and the algorithmic metatheorems presented in the talk lie at the heart of the proof.

---

## References

- 1 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.
- 2 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems*, 33(2):125–150, 2000. doi:10.1007/s002249910009.

- 3 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010)*, pages 143–152, 2010. doi:10.1109/FOCS.2010.21.
- 4 Markus Frick and Martin Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, November 2001. doi:10.1145/504794.504798.
- 5 Martin Grohe. Logic, graphs, and algorithms. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives*, volume 2 of *Texts in Logic and Games*, pages 357–422. Amsterdam University Press, 2007. URL: <http://www2.informatik.hu-berlin.de/~grohe/pub/meta-survey.pdf>.
- 6 Martin Grohe and Stephan Kreutzer. Methods for algorithmic meta theorems. In *Model Theoretic Methods in Finite Combinatorics*, volume 558 of *Contemporary Mathematics*, pages 181–206. American Mathematical Society, 2011. URL: <http://www2.informatik.hu-berlin.de/~grohe/pub/grokre11.pdf>.
- 7 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*, STOC’14, pages 89–98, New York, NY, USA, 2014. ACM. doi:10.1145/2591796.2591851.
- 8 Stephan Kreutzer. Algorithmic meta-theorems. In Javier Esparza, Christian Michaux, and Charles Steinhorn, editors, *Finite and Algorithmic Model Theory*, volume 379 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 2011. URL: <http://logic.las.tu-berlin.de/Kreutzer/Publications/amt-survey.pdf>.
- 9 Till Tantau. A gentle introduction to applications of algorithmic metatheorems for space and circuit classes. *Algorithms*, 9(3):44, 2016. URL: <http://www.mdpi.com/1999-4893/9/3/44>, doi:10.3390/a9030044.
- 10 Carsten Thomassen. On the presence of disjoint subgraphs of a specified type. *Journal of Graph Theory*, 12(1):101–111, 1988. doi:10.1002/jgt.3190120111.

# Split Contraction: The Untold Story\*

Akanksha Agrawal<sup>1</sup>, Daniel Lokshtanov<sup>2</sup>, Saket Saurabh<sup>3</sup>, and Meirav Zehavi<sup>4</sup>

1 Department of Informatics, University of Bergen, Bergen, Norway  
akanksha.agrawal@uib.no

1 Department of Informatics, University of Bergen, Bergen, Norway  
daniello@uib.no

3 Department of Informatics, University of Bergen, Bergen, Norway; and  
Institute of Mathematical Sciences, Chennai, India  
saket@imsc.res.in

4 Department of Informatics, University of Bergen, Bergen, Norway  
meirav.zehavi@uib.no

---

## Abstract

The edit operation that contracts edges, which is a fundamental operation in the theory of graph minors, has recently gained substantial scientific attention from the viewpoint of Parameterized Complexity. In this paper, we examine an important family of graphs, namely the family of split graphs, which in the context of edge contractions, is proven to be significantly less obedient than one might expect. Formally, given a graph  $G$  and an integer  $k$ , SPLIT CONTRACTION asks whether there exists  $X \subseteq E(G)$  such that  $G/X$  is a split graph and  $|X| \leq k$ . Here,  $G/X$  is the graph obtained from  $G$  by contracting edges in  $X$ . It was previously claimed that SPLIT CONTRACTION is fixed-parameter tractable. However, we show that SPLIT CONTRACTION, despite its deceptive simplicity, is W[1]-hard. Our main result establishes the following conditional lower bound: under the Exponential Time Hypothesis, SPLIT CONTRACTION cannot be solved in time  $2^{o(\ell^2)} \cdot n^{\mathcal{O}(1)}$  where  $\ell$  is the vertex cover number of the input graph. We also verify that this lower bound is essentially tight. To the best of our knowledge, this is the *first* tight lower bound of the form  $2^{o(\ell^2)} \cdot n^{\mathcal{O}(1)}$  for problems parameterized by the vertex cover number of the input graph. In particular, our approach to obtain this lower bound borrows the notion of harmonious coloring from Graph Theory, and might be of independent interest.

**1998 ACM Subject Classification** G.2.2 Graph Algorithms, I.1.2 Analysis of Algorithms

**Keywords and phrases** Split Graph, Parameterized Complexity, Edge Contraction

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.5

## 1 Introduction

Graph modification problems have been extensively studied since the inception of Parameterized Complexity in the early 90's. The input of a typical graph modification problem consists of a graph  $G$  and a positive integer  $k$ , and the objective is to edit  $k$  vertices (or edges) so that the resulting graph belongs to some particular family,  $\mathcal{F}$ , of graphs. These problems are not only mathematically and structurally challenging, but have also led to the discovery of several important techniques in the field of Parameterized Complexity. It would be completely appropriate to say that solutions to these problems played a central role in the growth of the

---

\* The research leading to these results has received funding from the European Research Council (ERC) via grant PARAPPROX, reference 306992.



field. In fact, just over the course of the last couple of years, parameterized algorithms have been developed for CHORDAL EDITING [9], UNIT INTERVAL EDITING [7], INTERVAL VERTEX (EDGE) DELETION [10, 8], PROPER INTERVAL COMPLETION [3], INTERVAL COMPLETION [4] CHORDAL COMPLETION [20], CLUSTER EDITING [19], THRESHOLD EDITING [16], CHAIN EDITING [16], TRIVIALY PERFECT EDITING [17, 15] and SPLIT EDITING [21]. This list is not comprehensive but rather illustrative.

The focus of all of these papers, and in fact, of the vast majority of papers on parameterized graph editing problems, has so far been limited to edit operations that delete vertices, delete edges or add edges. Using a different terminology, these problems can also be phrased as follows. For some particular family of graphs,  $\mathcal{F}$ , we say that a graph  $G$  belongs to  $\mathcal{F} + kv$ ,  $\mathcal{F} + ke$  or  $\mathcal{F} - ke$  if some graph in  $\mathcal{F}$  can be obtained by deleting at most  $k$  vertices from  $G$ , deleting at most  $k$  edges from  $G$  or adding at most  $k$  edges to  $G$ , respectively. Recently, a methodology for proving lower bounds on running times of algorithms for such parameterized graph editing problems was proposed by Bliznets et al. [2]. Furthermore, a well-known result by Cai [5] states that in case  $\mathcal{F}$  is a hereditary family of graphs with a finite set of forbidden induced subgraphs, then the graph modification problem defined by  $\mathcal{F}$  and the aforementioned edit operations admits a simple FPT algorithm.

In recent years, a different edit operation has begun to attract significant scientific attention. This operation, which is arguably the most natural edit operation apart from deletions/insertions of vertices/edges, is the one that contracts an edge. Here, given an edge  $(u, v)$  that exists in the input graph, we remove the edge from the graph and merge its two endpoints. Edge contraction is a fundamental operation in the theory of graph minors. Using our alternative terminology, we say that a graph  $G$  belongs to  $\mathcal{F}/ke$  if some graph in  $\mathcal{F}$  can be obtained by contracting at most  $k$  edges in  $G$ .<sup>1</sup> Then, given a graph  $G$  and a positive integer  $k$ ,  $\mathcal{F}$ -EDGE CONTRACTION asks whether  $G$  belongs to  $\mathcal{F}/ke$ . For several families of graphs  $\mathcal{F}$ , early papers by Watanabe et al. [34, 35] and Asano and Hirata [1] showed that  $\mathcal{F}$ -EDGE CONTRACTION is NP-complete. In the framework of Parameterized Complexity, these problems exhibit properties that are quite different from those of problems where we only delete or add vertices and edges. Indeed, for these problems, the result by Cai [5] does not hold. In particular, Lokshtanov et al. [31] and Cai and Guo [6] independently showed that if  $\mathcal{F}$  is either the family of  $P_\ell$ -free graphs for some  $\ell \geq 5$  or the family of  $C_\ell$ -free graphs for some  $\ell \geq 4$ , then  $\mathcal{F}$ -EDGE CONTRACTION is W[2]-hard.

To the best of our knowledge, Heggernes et al. [26] were the first to explicitly study  $\mathcal{F}$ -EDGE CONTRACTION from the viewpoint of Parameterized Complexity. They showed that in case  $\mathcal{F}$  is the family of trees,  $\mathcal{F}$ -EDGE CONTRACTION is FPT but does not admit a polynomial kernel, while in case  $\mathcal{F}$  is the family of paths, the corresponding problem admits a faster algorithm and an  $\mathcal{O}(k)$ -vertex kernel. Golovach et al. [22] proved that if  $\mathcal{F}$  is the family of planar graphs, then  $\mathcal{F}$ -EDGE CONTRACTION is again FPT. Moreover, Cai and Guo [6] showed that in case  $\mathcal{F}$  is the family of cliques,  $\mathcal{F}$ -EDGE CONTRACTION is solvable in time  $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ , while in case  $\mathcal{F}$  is the family of chordal graphs, the problem is W[2]-hard. Heggernes et al. [25] developed an FPT algorithm for the case where  $\mathcal{F}$  is the family of bipartite graphs. Later, a faster algorithm was proposed by Guillemot and Marx [23].

The recent paper [24] studied the case where  $\mathcal{F}$  is the family of split graphs, which corresponds to the following problem.

---

<sup>1</sup> Here, it might be more appropriate to replace  $/$  (in  $\mathcal{F}/ke$ ) by the operation opposite to edge contraction, but we believe that the current notation is clearer.

SPLIT CONTRACTION

Parameter:  $k$

**Input:** A graph  $G$  and an integer  $k$ .

**Question:** Does there exist  $X \subseteq E(G)$  such that  $G/X$  is a split graph and  $|X| \leq k$ ?

The paper [24] claimed to design an algorithm that solves SPLIT CONTRACTION in time  $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$ , which proves that the problem is FPT. Our initial objective was to either speed-up this algorithm or obtain a tight conditional lower bound. In fact, it seemed plausible that SPLIT CONTRACTION, like  $\mathcal{F}$ -EDGE CONTRACTION where  $\mathcal{F}$  is the family of cliques, is solvable in time  $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ . The algorithm in [24] first computes a set of vertices of small size whose removal renders the graph into a split graph. Then, it is based on case distinction. In case the remaining graph contains a large clique, the problem is solved in time  $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$ , and otherwise it is solved in time  $2^{\mathcal{O}(k^2)} \cdot n^{\mathcal{O}(1)}$ . In particular, in case the clique is small, the minimum size of a vertex cover of the input graph is small—it can be bounded by  $\mathcal{O}(k)$ . Thus, the bottleneck of the proposed algorithm is captured by graphs having small vertex covers. Interestingly, our first main result, given in Section 3, proves that it is unlikely to overcome the difficulty imposed by such graphs.

► **Theorem 1.** *Unless the ETH fails, SPLIT CONTRACTION parameterized by  $\ell$ , the size of a minimum vertex cover of the input graph, does not have an algorithm running in time  $2^{\mathcal{O}(\ell^2)} \cdot n^{\mathcal{O}(1)}$ . Here,  $n$  denotes the number of vertices in the input graph.*

To the best of our knowledge, under the Exponential Time Hypothesis (ETH) [12, 27], this is the *first* tight lower bound of this form for problems parameterized by the vertex cover number of the input graph. Lately, there has been increasing scientific interest in the examination of lower bounds of forms other than  $2^{o(s)} \cdot n^{\mathcal{O}(1)}$  for some parameters  $s$ . For example, lower bounds that are “slightly super-exponential”, i.e. of the form  $2^{o(s \log s)} \cdot n^{\mathcal{O}(1)}$  for various parameters  $s$ , have been studied in [30]. Cygan et al. [13] obtained a lower bound of the form  $2^{2^{o(k)}} \cdot n^{\mathcal{O}(1)}$ , where  $k$  is the solution size, for the EDGE CLIQUE COVER problem. Very recently, Marx and Mitsoué [32] have further obtained lower bounds of the forms  $2^{2^{o(w)}} \cdot n^{\mathcal{O}(1)}$  and  $2^{2^{2^{o(w)}}} \cdot n^{\mathcal{O}(1)}$ , where  $w$  is the treewidth of the input graph, for choosability problems.

In order to derive our main result, we make use of a partitioning of the vertex set  $V(G)$  into independent sets  $C_1, \dots, C_t$  such that for each  $i, j \in [t]$ ,  $i \neq j$ ,  $|E(G[C_i \cup C_j]) \cap E(G)| \leq 1$ . Essentially, this coloring can be viewed as a proper coloring  $f : V(G) \rightarrow [t]$  with the additional property that between any two color classes we have at most one edge. (Here, we use the standard notation  $[t] = \{1, 2, \dots, t\}$ .) This kind of coloring, called *harmonious coloring* [29, 33, 18], has been studied extensively in the literature. We are not aware of uses of harmonious coloring in deriving lower bound results and believe that this approach could be of independent interest.

After we had established Theorem 1, we took a closer look at the paper [24], and were not able to verify some of their arguments. We next prove that unless  $\text{FPT} = \text{W}[1]$ -hard, the algorithm in [24] is incorrect, as the problem is  $\text{W}[1]$ -hard (Section 4).

► **Theorem 2.** *SPLIT CONTRACTION parameterized by the size of a solution is  $\text{W}[1]$ -hard.*

We find this result surprising: one might a priori expect that “contraction to split graphs” should be easy as split graphs have structures that seem relatively simple. Indeed, many NP-hard problems admit simple polynomial-time algorithms if restricted to split graphs. Consequently, our result can also be viewed as a strong evidence of the inherent complexity of the edit operation which contracts edges. Furthermore, some of the ideas underlying the

constructions of this reduction, such as the exploitation of properties of a special case of the PERFECT CODE problem to analyze budget constraints involving edge contractions, might be used to establish other W[1]-hard results for problems of similar flavors. We remark that despite errors in the paper [24], it can be verified that the lower bound given by Theorem 1 is tight. For the sake of completeness, we design a standalone FPT algorithm for SPLIT CONTRACTION that runs in time  $2^{\mathcal{O}(\ell^2)} \cdot n^{\mathcal{O}(1)}$ , the details of which are omitted due to space constraints.

## 2 Preliminaries

We consider only finite simple graphs. A *split graph* is a graph  $G$  whose vertex set  $V(G)$  can be partitioned into two sets,  $A$  and  $B$ , such that  $G[A]$  is a clique while  $B$  is an independent set, i.e.  $G[B]$  is an edgeless graph. We say that two disjoint vertex subsets, say  $S, S' \subseteq V(G)$ , are *adjacent* if there exist  $v \in S$  and  $v' \in S'$  such that  $(v, v') \in E(G)$ . Further, an edge  $(u, v) \in E(G)$  is *between*  $S, S'$  if  $u \in S$  and  $v \in S'$  (or  $v \in S$  and  $u \in S'$ ). For  $(v, u) \in E(G)$ , the result of *contracting* the edge  $(v, u)$  in  $G$  is the graph obtained by the following operation. We add a vertex  $vu^*$  and make it adjacent to the vertices in  $(N(v) \cup N(u)) \setminus \{v, u\}$ , and delete  $v, u$  from the graph. We often call such an operation a *contraction* of the edge  $(v, u)$ . For  $E' \subseteq E(G)$ , we denote by  $G/E'$  the graph obtained by contracting the edges of  $E'$  in  $G$ .

A graph  $G$  is *isomorphic* to a graph  $H$  if there exists a *bijective* function  $\phi : V(G) \rightarrow V(H)$  such that for  $v, u \in V(G)$ ,  $(v, u) \in E(G)$  if and only if  $(\phi(v), \phi(u)) \in E(H)$ . A graph  $G$  is *contractible* to a graph  $H$  if there exists  $E' \subseteq E(G)$  such that  $G/E'$  is isomorphic to  $H$ . In other words,  $G$  is contractible to  $H$  if there exists a *surjective* function  $\varphi : V(G) \rightarrow V(H)$  with the following properties.

- For all  $h, h' \in V(H)$ ,  $(h, h') \in E(H)$  if and only if  $W(h), W(h')$  are *adjacent* in  $G$ . Here,  $W(h) = \{v \in V(G) \mid \varphi(v) = h\}$ .
- For all  $h \in V(H)$ ,  $G[W(h)]$  is *connected*.

Let  $\mathcal{W} = \{W(h) \mid h \in V(H)\}$ . Observe that  $\mathcal{W}$  defines a partition of the vertex set of  $G$ . We call  $\mathcal{W}$  a *H-witness structure* of  $G$ . The sets in  $\mathcal{W}$  are called *witness-sets*.

## 3 Lower Bound for Split-Contraction Parameterized by Vertex Cover

In this section we show that unless the ETH fails, SPLIT CONTRACTION does not admit an algorithm running in time  $2^{\mathcal{O}(\ell^2)} n^{\mathcal{O}(1)}$ , where  $\ell$  is the size of a minimum vertex cover of the input graph  $G$  on  $n$  vertices. We complement it by designing an algorithm (whose details are omitted) for SPLIT CONTRACTION parameterized by  $\ell$ , running in time  $2^{\mathcal{O}(\ell^2)} n^{\mathcal{O}(1)}$ .

To obtain our lower bound, we give an appropriate reduction from VERTEX COVER on sub-cubic graphs. For this we utilize the fact that VERTEX COVER on sub-cubic graphs does not have an algorithm running in time  $2^{\mathcal{O}(n)} n^{\mathcal{O}(1)}$  unless the ETH fails [27, 28]. For the ease of presentation we split the reduction into two steps. The first step comprises of reducing a special case of VERTEX COVER on sub-cubic graphs, which we call SUB-CUBIC PARTITIONED VERTEX COVER (SUB-CUBIC PVC) to SPLIT CONTRACTION. In the second step we show that there does not exist an algorithm for SUB-CUBIC PVC running in time  $2^{\mathcal{O}(n)} n^{\mathcal{O}(1)}$  for SUB-CUBIC PVC. We remark that the reduction from VERTEX COVER on sub-cubic graphs to SUB-CUBIC PVC is a Turing reduction.



### 3.1 Reduction from Sub-Cubic Partitioned Vertex Cover to Split Contraction

In this section we give a reduction from SUB-CUBIC PARTITIONED VERTEX COVER to SPLIT CONTRACTION. Next we formally define SUB-CUBIC PARTITIONED VERTEX COVER.

SUB-CUBIC PARTITIONED VERTEX COVER (SUB-CUBIC PVC)

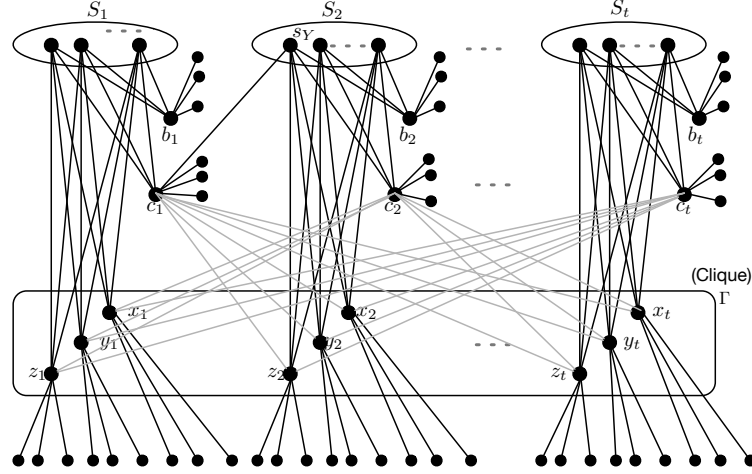
**Input:** A sub-cubic graph  $G$ ; an integer  $t$ ; for  $i \in [t]$ , an integer  $k_i \geq 0$ ; a partition  $\mathcal{P} = \{C_1, \dots, C_t\}$  of  $V(G)$  such that  $t \in \mathcal{O}(\sqrt{|V(G)|})$  and for all  $i \in [t]$ ,  $C_i$  is an independent set and  $|C_i| \in \mathcal{O}(\sqrt{|V(G)|})$ . Furthermore, for  $i, j \in [t], i \neq j$ ,  $|E(G[C_i \cup C_j]) \cap E(G)| = 1$ .

**Question:** Does  $G$  have a vertex cover  $X$  such that for all  $i \in [t]$ ,  $|X \cap C_i| \leq k_i$ ?

We first explain (informally) the ideas behind our reduction. Let  $X$  be a *hypothetical* vertex cover we are looking for. Recall that we assume the ETH holds and thus we are allowed to use  $2^{o(n)}n^{\mathcal{O}(1)}$  time to obtain our reduction. We will use this freedom to design our reduction and to construct an instance  $(G', k')$  of SPLIT CONTRACTION. For  $i \in [t]$ , in  $V(G')$ , we have a *vertex* corresponding to each possible intersection of  $X$  with  $C_i$  on at most  $k_i$  vertices. Further, we have a vertex  $c_i \in V(G')$  corresponding to each  $C_i$ , for  $i \in [t]$ . We want to make sure that for each  $(u, v) \in E(G)$ , we choose an edge of  $E(G')$  (in the solution to SPLIT CONTRACTION) that is incident to a vertex which corresponds to a subset containing one of  $u$  or  $v$  and one of  $c_i$  or  $c_j$ . Furthermore, we want to force these selected vertices to be contracted to the clique side in the resulting split graph. We crucially exploit the fact that there is exactly one edge between every  $C_i, C_j$  pair, where  $i, j \in [t], i \neq j$ . Finally, we will add a clique, say  $\Gamma$ , of size  $3t$  and make each of its vertices adjacent to many pendant vertices, which ensures that after contracting the solution edges, the vertices of  $\Gamma$  remain in the clique side. We will assign appropriate adjacencies between the vertices of  $\Gamma$  and  $c_i$ , for  $i \in [t]$ . This will guide us in selecting edges for the solution of the contraction problem. We now move to the formal description of the construction used in the reduction.

**Construction.** Let  $(G, \mathcal{P} = \{C_1, C_2, \dots, C_t\}, k_1, \dots, k_t)$  be an instance of SUB-CUBIC PVC and  $n = |V(G)|$ . We create an instance of SPLIT CONTRACTION  $(G', k')$  as follows. For  $i \in [t]$ , let  $S_i = \{v_Y \mid Y \subseteq C_i \text{ and } |Y| \leq k_i\}$ . That is,  $S_i$  comprises of vertices corresponding to subsets of  $C_i$  of size at most  $k_i$ . For each  $i \in [t]$ , we add five vertices  $b_i, c_i, x_i, y_i, z_i$  to  $V(G')$ . The vertices  $\{x_i, y_i, z_i \mid i \in [t]\}$  induce a clique (on  $3t$  vertices) in  $G'$ . We add the edges  $(b_i, s_Y), (c_i, s_Y), (x_i, s_Y), (y_i, s_Y), (z_i, s_Y)$  for all  $s_Y \in S_i$  to  $E(G')$ . For  $i, j \in [t], i \neq j$ , we add the edges  $(c_i, x_j), (c_i, y_j), (c_i, z_j)$  to  $E(G')$ . For  $i, j \in [t], i \neq j$  and  $s_Y \in S_j$ , we add the edge  $(c_i, s_Y)$  in  $E(G')$  if and only if  $Y$  covers the unique edge between  $C_i$  and  $C_j$ . For all  $i \in [t]$ , we add  $4t + 2$  pendant vertices,  $b_j^i, j \in [4t + 2]$ , to  $b_i$ . Similarly, for all  $i \in [t]$ , we add  $4t + 2$  pendant vertices  $c_j^i, x_j^i, y_j^i$ , and  $z_j^i, j \in [4t + 2]$ , to  $c_i, x_i, y_i$  and  $z_i$ , respectively. The pendant vertices are added in order to make sure that the vertices resulting after the contraction of their witness sets belong to the clique side. This completes the construction of the graph  $G'$ . Observe that  $\{b_i, c_i, x_i, y_i, z_i \mid i \in [t]\}$  forms a minimum vertex cover of  $G'$  of size  $5t$ . Finally, we set  $k' = 2t$ . The resulting instance of SPLIT CONTRACTION is  $(G', k')$ . We refer the reader to Figure 1 for an illustration of the construction.

In the next few lemmata (Lemmata 3 to 8) we prove certain properties of the instance  $(G', k')$  of SPLIT CONTRACTION. This will be helpful later for establishing the equivalence between the original instance  $(G, \mathcal{P} = \{C_1, C_2, \dots, C_t\}, k_1, \dots, k_t)$  of SUB-CUBIC PVC and the instance  $(G', k')$  of SPLIT CONTRACTION. In Lemmas 3 to 8 we will use the following notations. We use  $T$  to denote a solution to SPLIT CONTRACTION in  $(G', k')$  and  $H = G'/T$



■ **Figure 1** Reduction from SUB-CUBIC PVC to SPLIT CONTRACTION.

with  $\hat{C}, \hat{I}$  being a partition of  $V(H)$  inducing a clique and an independent set, respectively, in  $H$ . We let  $\varphi : V(G') \rightarrow V(H)$  be the surjective function defining the contractibility of  $G'$  to  $H$ , and  $\mathcal{W}$  be the  $H$ -witness structure of  $G'$ .

► **Lemma 3.** *Let  $(G', k')$  be a YES instance of SPLIT CONTRACTION. Then, for all  $v \in \{b_i, c_i, x_i, y_i, z_i \mid i \in [t]\}$ , we have  $\varphi(v) \in \hat{C}$ .*

**Proof.** Consider  $v \in \{b_i, c_i, x_i, y_i, z_i \mid i \in [t]\}$ . Recall that there are  $4t + 2 = 2k' + 2$  pendant vertices  $v_j^i$ , for  $j \in [2k' + 2]$  adjacent to  $v$ . At most  $k'$  edges in  $\{(v_j^i, v) \mid j \in [2k' + 2]\}$  can belong to  $T$ . Therefore, there exist  $j_1, j_2 \in [2k' + 2]$ ,  $j_1 \neq j_2$  such that no edge incident to  $v_{j_1}^i$  or  $v_{j_2}^i$  is in  $T$ . In other words, for  $h_1 = \varphi(v_{j_1}^i)$  and  $h_2 = \varphi(v_{j_2}^i)$ ,  $W(h_1)$  and  $W(h_2)$  are singleton sets. Since  $\mathcal{W}$  is a  $H$ -witness structure of  $G'$ ,  $(h_1, h_2) \notin E(H)$ . Therefore, at least one of  $h_1, h_2$  belongs to  $\hat{I}$ , say  $h_1 \in \hat{I}$ . This implies that  $\varphi(v) \in \hat{C}$ . ◀

► **Lemma 4.** *Let  $(G', k')$  be a YES instance of SPLIT CONTRACTION. Then, for all  $i \in [t]$ , there exists  $s_{Y_i} \in S_i$  such that  $(b_i, s_{Y_i}) \in T$ .*

**Proof.** Towards a contradiction assume that there is  $i \in [t]$  such that for all  $s_{Y_i} \in S_i$ ,  $(b_i, s_{Y_i}) \notin T$ . Recall that  $N_{G'}(b_i) = S_i \cup \{b_j^i \mid j \in [4t + 2]\}$ . Let  $h = \varphi(b_i)$  and  $A = \{b_j, c_j, x_j, y_j, z_j \mid j \in [t], j \neq i\}$ . There exists  $v \in A$  such that  $|W(h')| = 1$ , where  $h' = \varphi(v)$ . This follows from the fact that at most  $2k' = 4t$  vertices in  $A$  can be incident to an edge in  $T$ , although  $|A| = 5(t - 1) > 4t$ , as  $t$  can be assumed to be larger than 6, else the graph has constantly many edges and we can solve the problem in polynomial time. From Lemma 3 it follows that  $(h, h') \in E(H)$ , but  $W(h), W(h')$  are not adjacent in  $G'$ , contradicting that  $\mathcal{W}$  is an  $H$ -witness structure of  $G'$ . Hence the claim follows. ◀

For each  $i \in [t]$ , we arbitrarily choose a vertex  $s_{Y_i}^* \in S_i$  such that  $(b_i, s_{Y_i}^*) \in T$ . The existence of such a vertex is guaranteed by Lemma 4.

► **Lemma 5.** *Let  $(G', k')$  be a YES instance of SPLIT CONTRACTION and  $(b_i, s_{Y_i}^*) \in T$  for  $i \in [t]$ . Then, for  $h_i = \varphi(s_{Y_i}^*)$ , we have  $|W(h_i)| \geq 3$ . Furthermore, there is an edge in  $T$  incident to  $b_i$  or  $s_{Y_i}^*$  other than  $(b_i, s_{Y_i}^*)$ .*

**Proof.** Suppose there exists  $i \in [t]$ ,  $h_i = \varphi(s_{Y_i}^*)$  such that  $|W(h_i)| < 3$ . Recall that  $|W(h_i)| \geq 2$ , since  $b_i \in W(h_i)$ . Let  $A = \{x_j, y_j, z_j \mid j \in [t], j \neq i\}$ . From Lemma 4, it

follows that for each  $j \in [t]$ , there is an edge  $(b_j, s_{Y_j}^*) \in T$ , therefore the number of edges in  $T$  incident to a vertex in  $A$  is bounded by  $k' - t = t$ . But  $|A| = 3t - 3 > 2t$ , therefore, there exists  $a \in A$  such that for  $h_a = \varphi(a)$ ,  $|W(h_a)| = 1$ . From Lemma 3,  $(h_i, h_a) \in E(H)$ , therefore  $W(h_i)$  and  $W(h_a)$  must be adjacent in  $G'$ . But  $a \notin N(\{b_i, s_{Y_i}^*\})$ , hence  $W(h_i)$  and  $W(h_a)$  are not adjacent in  $G'$ , contradicting that  $\mathcal{W}$  is an  $H$ -witness structure of  $G'$ .

Since  $|W(h_i)| \geq 3$  and  $G[W(h_i)]$  is connected, at least one of  $s_{Y_i}^*, b_i$  must be adjacent to an edge in  $T$  which is not  $(s_{Y_i}^*, b_i)$ .  $\blacktriangleleft$

► **Lemma 6.** *Let  $(G', k')$  be a YES instance of SPLIT CONTRACTION. Then, for all  $i \in [t]$ , we have  $|W(h_i)| \geq 2$  where  $h_i = \varphi(c_i)$ .*

**Proof.** Towards a contradiction assume that there exists  $i \in [t]$ ,  $h_i = \varphi(c_i)$ , such that  $|W(h_i)| < 2$ . Let  $A = \{c_j \mid j \in [t], j \neq i\} \cup \{x_i, y_i, z_i\}$ . From Lemma 4 it follows that the edge  $(b_j, s_{Y_j}^*) \in T$ , for each  $j \in [t]$ . By Lemma 5 it follows that there is an edge in  $T$  that is adjacent to exactly one of  $\{b_j, s_{Y_j}^*\}$  in  $T$ , for all  $j \in [t]$ . Therefore, at most  $t$  vertices in  $A$  can be adjacent to an edge in  $T$ , while  $|A| = t + 2$ . This implies that there exists  $a \in A$ ,  $h_a = \varphi(a)$  such that  $|W(h_a)| = 1$ . Observe that none of the vertices in  $A$  are adjacent to  $c_i$  in  $G'$ . Therefore, it follows that  $W(h_i), W(h_a)$  are not adjacent in  $G'$ . But Lemma 3 implies that  $(h_i, h_a) \in E(H)$ , a contradiction to  $\mathcal{W}$  being an  $H$ -witness structure of  $G'$ .  $\blacktriangleleft$

► **Lemma 7.** *Let  $(G', k')$  be a YES instance of SPLIT CONTRACTION and  $(b_i, s_{Y_i}^*) \in T$  for  $i \in [t]$ . Then, for each  $i \in [t]$ , we have  $|W(h_i)| = 3$  where  $h_i = \varphi(s_{Y_i}^*)$ .*

**Proof.** For  $i \in [t]$ , let  $h_i = \varphi(s_{Y_i}^*)$ . From Lemma 5 we know that  $|W(h_i)| \geq 3$ . Let  $C = \{c_i \mid i \in [t]\}$  and  $\mathcal{S} = \{(b_i, s_{Y_i}^*) \mid i \in [t]\}$ . From Lemmata 5 and 6 it follows that each  $c \in C$  must be incident to an edge in  $T$  and each  $S \in \mathcal{S}$  must have a vertex which is incident to an edge in  $T$  with the other endpoint not in  $S$ . Since  $|C| = |\mathcal{S}| = t$  and  $(b_i, s_{Y_i}^*) \in T$ , for all  $i \in [t]$ , there are at most  $t$  edges in  $T$  that are incident to a vertex in  $C$  and a vertex in  $S \in \mathcal{S}$ . Therefore, each  $c \in C$  is incident to exactly one edge in  $T$ . Similarly, each  $S \in \mathcal{S}$  is incident to exactly one edge with one endpoint in  $S$  and the other not in  $S$ . This implies that exactly one vertex  $c \in C$  belongs to  $W(h_i)$  for  $i \in [t]$ , and  $c$  does not belong to  $W(h_j)$ , where  $i \neq j$ ,  $i, j \in [t]$ . Also note that none of the vertices in  $\{x_i, y_i, z_i \mid i \in [t]\}$  can be incident to an edge in  $T$ . Similarly, none of the vertices in  $\{b_j^i, c_j^i, x_j^i, y_j^i, z_j^i \mid i \in [t], j \in [4t + 2]\}$  can be incident to an edge in  $T$ . Hence, we get that  $|W(h_i)| = 3$ , concluding the proof.  $\blacktriangleleft$

► **Lemma 8.** *Let  $(G', k')$  be a YES instance of SPLIT CONTRACTION and  $(b_i, s_{Y_i}^*) \in T$  for  $i \in [t]$ . Then, for all  $i \in [t]$ , we have  $c_i \in W(h_i)$  where  $h_i = \varphi(s_{Y_i}^*)$ .*

**Proof.** Suppose for some  $i \in [t]$ ,  $c_i \notin W(h_i)$  where  $h_i = \varphi(s_{Y_i}^*)$ . From Lemmata 5, 6 and  $k' = 2t$ , it follows that there exists some  $j \in [t]$  such that  $c_i \in W(h_j)$ , where  $h_j = \varphi(s_{Y_j}^*)$ . By our assumption,  $j \neq i$ . From Lemma 7 we know that  $|W(h_j)| = 3$ , therefore  $W(h_j) = \{b_j, s_{Y_j}^*, c_i\}$ . Moreover, by Lemmata 6 and 7 and since  $k' = 2t$ ,  $|W(x_i)| = 1$ . However, we then get that  $W(h_j), W(x_i)$  are not adjacent in  $G'$ . By Lemma 3, we obtain a contradiction to the assumption that  $\mathcal{W}$  is an  $H$ -witness structure of  $G'$ . This completes the proof.  $\blacktriangleleft$

We are now ready to prove the main equivalence lemma of this section.

► **Lemma 9.**  *$(G, \mathcal{P} = \{C_1, C_2, \dots, C_t\}, k_1, \dots, k_t)$  is a YES instance of SUB-CUBIC PVC if and only if  $(G', k')$  is a YES instance of SPLIT CONTRACTION.*

**Proof.** In the forward direction, let  $Y$  be a vertex cover in  $G$  such that for each  $i \in [t]$ ,  $|Y \cap C_i| \leq k_i$ . For  $i \in [t]$ , we let  $Y_i = Y \cap C_i$ . Let  $T = \{(b_i, s_{Y_i}), (c_i, s_{Y_i}) \mid i \in [t]\}$ . Let

$H = G'/T$ ,  $\varphi : V(G') \rightarrow V(H)$  be the underlying surjective map and  $\mathcal{W}$  be the  $H$ -witness structure of  $G'$ . To show that  $T$  is a solution to SPLIT CONTRACTION in  $(G', k')$ , it is enough to show that  $H$  is a split graph. Let  $I = \cup_{i \in [t]} (S_i \setminus \{s_{Y_i}\}) \cup \{b_j^i, c_j^i, x_j^i, y_j^i, z_j^i \mid i \in [t], j \in [4t+2]\}$ . Recall that for each  $v \in I$ ,  $|W(\varphi(v))| = 1$ . Furthermore, for  $v, v' \in I$ ,  $(v, v') \notin E(G')$ . Hence, it follows that  $\hat{I} = \{\varphi(v) \mid v \in I\}$  induces an independent set in  $H$ . Let  $\mathcal{C}_1 = \{x_i, y_i, z_i \mid i \in [t]\}$ . Recall that  $G'[\mathcal{C}_1]$  is a clique and from the construction of  $T$ ,  $|W(\varphi(c))| = 1$  for all  $c \in \mathcal{C}_1$ . Therefore,  $\hat{\mathcal{C}}_1 = \{\varphi(c) \mid c \in \mathcal{C}_1\}$  induces a clique in  $H$ . Let  $\mathcal{C}_2 = \{s_{Y_i} \mid i \in [t]\}$ ,  $h_i = \varphi(s_{Y_i})$  for  $i \in [t]$ , and  $\hat{\mathcal{C}}_2 = \{h_i \mid i \in [t]\}$ . From the construction of  $T$ , we have  $W(h_i) = \{b_i, c_i, s_{Y_i}\}$  for all  $i \in [t]$ . Observe that for  $c_1 \in \hat{\mathcal{C}}_1$  and  $c_2 \in \hat{\mathcal{C}}_2$ ,  $W(c_1), W(c_2)$  are adjacent in  $G'$ , therefore,  $(c_1, c_2) \in E(H)$ . Consider  $h_i, h_j \in \hat{\mathcal{C}}_2$ , where  $i, j \in [t], i \neq j$ . Recall  $W(h_i) = \{b_i, s_{Y_i}, c_i\}$  and  $W(h_j) = \{b_j, s_{Y_j}, c_j\}$ . Since  $Y$  is a vertex cover, at least one of  $Y_i$  or  $Y_j$  covers the unique edge between  $C_i$  and  $C_j$  in  $G$ , say  $Y_i$  covers the edge between  $C_i$  and  $C_j$ . But then  $(s_{Y_i}, c_j) \in E(G')$ , therefore  $(h_i, h_j) \in E(H)$ . The above argument implies that  $\hat{\mathcal{C}} = \hat{\mathcal{C}}_1 \cup \hat{\mathcal{C}}_2$  induces a clique in  $H$ . Furthermore,  $V(H) = \hat{I} \cup \hat{\mathcal{C}}$ . This implies that  $H$  is a split graph.

In the reverse direction, let  $T$  be a solution to SPLIT CONTRACTION in  $(G', k')$ . Let  $H = G'/T$ ,  $\varphi : V(G') \rightarrow V(H)$  be the underlying surjective map and  $\mathcal{W}$  be the  $H$ -witness structure of  $G'$ . From Lemma 4, it follows that for all  $i \in [t]$ , there exists  $s_{Y_i} \in S_i$  such that  $(b_i, s_{Y_i}) \in T$ . For  $i \in [t]$ , let  $Y_i$  be the set such that  $(b_i, s_{Y_i}) \in T$ . We let  $Y = \cup_{i \in [t]} Y_i$ . For  $i \in [t]$ , from the definition of the vertices in  $S_i$ , it follows that  $|Y \cap C_i| \leq k_i$ . We will show that  $Y$  is a vertex cover in  $G$ . Towards a contradiction assume that there exists  $i, j \in [t], i \neq j$ , such that  $Y$  does not cover the unique edge between  $C_i$  and  $C_j$ . From Lemmas 4 and 8 it follows that  $W(h_i) = \{b_i, s_{Y_i}, c_i\}$  and  $W(h_j) = \{b_j, s_{Y_j}, c_j\}$ , where  $h_i = \varphi(s_{Y_i})$  and  $h_j = \varphi(s_{Y_j})$ . From Lemma 3 it follows that  $(h_i, h_j) \in E(H)$ . Therefore,  $W(h_i)$  and  $W(h_j)$  are adjacent in  $G'$ . Recall that  $N_{G'}(b_i) \cap W(h_j) = \emptyset$ ,  $N_{G'}(b_j) \cap W(h_i) = \emptyset$ ,  $(c_i, c_j), (s_{Y_i}, s_{Y_j}) \notin E(G')$ . Therefore, at least one of  $(c_i, s_{Y_j}), (c_j, s_{Y_i})$  must belong to  $E(G')$ , say  $(c_i, s_{Y_j}) \in E(G')$ . But then by construction it follows that  $Y_j \subseteq Y$  covers the unique edge between  $C_i$  and  $C_j$  in  $G$ , a contradiction. This completes the proof.  $\blacktriangleleft$

Finally, we restate Theorem 1 and prove its correctness.

**► Theorem 10.** *Unless the ETH fails, SPLIT CONTRACTION parameterized by  $\ell$ , the size of a minimum vertex cover of the input graph, does not have an algorithm running in time  $2^{o(\ell^2)} \cdot n^{\mathcal{O}(1)}$ . Here,  $n$  denotes the number of vertices in the input graph.*

**Proof.** Towards a contradiction assume that there is an algorithm  $\mathcal{A}$  for SPLIT CONTRACTION, parameterized by  $\ell$ , the size of a minimum vertex cover, running in time  $2^{o(\ell^2)} n^{\mathcal{O}(1)}$ . Let  $(G, \mathcal{P} = \{C_1, C_2, \dots, C_t\}, k_1, \dots, k_t)$  be an instance of SUB-CUBIC PVC. We create an instance  $(G', k')$  of SPLIT CONTRACTION as described in the **Construction**, running in time  $2^{o(n)} \cdot n^{\mathcal{O}(1)}$ , where  $n = |V(G)|$ . Recall that in the instance created, the size of a minimum vertex cover is  $\ell = 5t = \mathcal{O}(\sqrt{n})$ . Then we use algorithm  $\mathcal{A}$  for deciding if  $(G', k')$  is a YES instance of SPLIT CONTRACTION and return the same answer for SUB-CUBIC PVC on  $(G, \mathcal{P}, k_1, \dots, k_t)$ . The correctness of the answer returned follows from Lemma 9. But then we can decide whether  $(G, \mathcal{P}, k_1, \dots, k_t)$  is a YES instance of SUB-CUBIC PVC in time  $2^{o(n)} \cdot n^{\mathcal{O}(1)}$ , which contradicts ETH assuming Theorem 11. This concludes the proof.  $\blacktriangleleft$

### 3.2 Reduction from Sub-Cubic VC to Sub-Cubic PVC

Finally, to complete our proof we show that SUB-CUBIC PVC on graphs with  $n$  vertices can not be solved in time  $2^{o(n)} n^{\mathcal{O}(1)}$  unless the ETH fails. In this section we give a Turing reduction from SUB-CUBIC VC to SUB-CUBIC PVC that will imply our desired assertion.

Let  $(G, k)$  be an instance of SUB-CUBIC VC and  $n = |V(G)|$ . We first create a new instance  $(G', k')$  of SUB-CUBIC VC satisfying certain properties. We start by computing a harmonious coloring of  $G$  using  $t \in \mathcal{O}(\sqrt{n})$  color classes such that each color class contains at most  $\mathcal{O}(\sqrt{n})$  vertices. A harmonious coloring on bounded degree graphs can be computed in polynomial time using at most  $\mathcal{O}(\sqrt{n})$  colors with each color class having at most  $\mathcal{O}(\sqrt{n})$  vertices [29, 33, 18]. Let  $C_1, \dots, C_t$  be the color classes. Recall that between each pair of the color classes,  $C_i, C_j$  for  $i, j \in [t], i \neq j$ , we have at most one edge. If for some  $i, j \in [t], i \neq j$ , there is no edge between a vertex in  $C_i$  and a vertex in  $C_j$ , then we add a new vertex  $x_{ij}$  in  $C_i$  and a new vertex  $x_{ji}$  in  $C_j$  and add the edge  $(x_{ij}, x_{ji})$ . Observe that we add a matching corresponding to a missing edge between a pair of color classes. In this process we can add at most  $t - 1$  new vertices to a color class  $C_i$ , for  $i \in [t]$ . Therefore, the number of vertices in  $C_i$  for  $i \in [t]$  after addition of new vertices is also bounded by  $\mathcal{O}(\sqrt{n})$ . We denote the resulting graph by  $G'$  with partition of vertices  $C_1, \dots, C_t$  (including the newly added vertices, if any). Observe that the number of vertices  $n'$  in  $G'$  is at most  $\mathcal{O}(n)$ . Let  $m$  be the number of matching edges added in  $G$  to obtain  $G'$  and let  $k' = k + m$ . It is easy to see that  $(G, k)$  is a YES instance of SUB-CUBIC VC if and only if  $(G', k')$  is a yes instance of SUB-CUBIC VC.

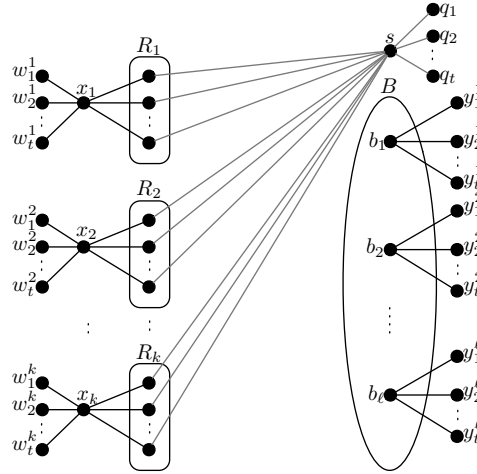
We will now be working with the instance  $(G', k')$  of SUB-CUBIC VC with the partition of vertices  $C_1, \dots, C_t$  obtained by extending the color classes of the harmonious coloring of  $G$  we started with. We guess the size of the intersection of the vertex cover in  $G'$  with each  $C_i$ , for  $i \in [t]$ . That is, for  $i \in [t]$ , we guess an integer  $0 \leq k'_i \leq \min(|C_i|, k')$ , such that  $\sum_{i \in [t]} k'_i = k'$ . Finally, we let  $(G', \mathcal{P} = \{C_1, \dots, C_t\}, k'_1, \dots, k'_t)$  be an instance of SUB-CUBIC PVC. Notice that  $G'$  and  $\mathcal{P}$  satisfies all the requirements for it to be an instance of SUB-CUBIC PVC. It is easy to see that  $(G', k')$  is a YES instance of SUB-CUBIC VC if and only if for some guess of  $k_i$ , for  $i \in [t]$ ,  $(G', \mathcal{P} = \{C_1, \dots, C_t\}, k'_1, \dots, k'_t)$  is a YES instance of SUB-CUBIC PVC. This finishes the reduction from SUB-CUBIC VC to SUB-CUBIC PVC.

► **Theorem 11.** *Unless the ETH fails, SUB-CUBIC PVC does not admit an algorithm running in time  $2^{o(n)} \cdot n^{\mathcal{O}(1)}$ . Here,  $n$  is the number of vertices in the input graph.*

**Proof.** Towards a contradiction assume that there is an algorithm  $\mathcal{A}$  for SUB-CUBIC PVC running in time  $2^{o(n)} \cdot n^{\mathcal{O}(1)}$ . Let  $(G, k)$  be an instance of SUB-CUBIC VC. We apply the above mentioned reduction to create an instance  $(G', k')$  of SUB-CUBIC VC with vertex partitions  $C_1, \dots, C_t$  such that  $t \in \mathcal{O}(\sqrt{n})$  and  $|C_i| \in \mathcal{O}(\sqrt{n})$ , for all  $i \in [t]$ . Furthermore, there is exactly one edge between  $C_i, C_j$ , for  $i, j \in [t], i \neq j$ , and  $C_i$  induces an independent set in  $G'$ . For each guess  $0 \leq k'_i \leq \min(|C_i|, k')$  of the size of intersection of vertex cover with  $C_i$ , for  $i \in [t]$ , we solve the instance  $(G', \mathcal{P}, k'_1, \dots, k'_t)$ . By the exhaustiveness of the guesses of the size of intersection for each partition,  $(G', k')$  is a YES instance of SUB-CUBIC VC if and only if for some guess  $k'_1, \dots, k'_t$ ,  $(G', \mathcal{P}, k'_1, \dots, k'_t)$  is a YES instance of SUB-CUBIC PVC. We emphasize the fact that the number of guesses we make is bounded by  $\sqrt{n}^{\mathcal{O}(\sqrt{n})} = 2^{o(n)}$ , since  $|C_i| \in \mathcal{O}(\sqrt{n})$  and  $t \in \mathcal{O}(\sqrt{n})$ . But then we have an algorithm for SUB-CUBIC VC running in time  $2^{o(n)} \cdot n^{\mathcal{O}(1)}$ , contradicting the ETH. This concludes the proof. ◀

## 4 W[1]-Hardness of Split Contraction

In this section we show that SPLIT CONTRACTION parameterized by the solution size is W[1]-hard. Towards this we first define an intermediate problem from which we give the desired reduction.



■ **Figure 2** W[1]-Hardness of SPLIT CONTRACTION.

SPECIAL RED-BLUE PERFECT CODE (SRBPC) **Parameter:**  $k$

**Input:** A bipartite graph  $G$  with vertex set  $V(G)$  partitioned into  $\mathcal{R}$  (red set) and  $\mathcal{B}$  (blue set). Furthermore,  $\mathcal{R}$  is partitioned (disjoint) into  $R_1 \uplus R_2 \uplus \dots \uplus R_k$  and for all  $r, r' \in \mathcal{R}$ ,  $d_{\mathcal{B}}(r) = d_{\mathcal{B}}(r')$ . That is, every vertex in  $\mathcal{R}$  has same degree, say  $d$ .

**Question:** Does there exist  $X \subseteq \mathcal{R}$ , such that for all  $b \in \mathcal{B}$ ,  $|N(b) \cap X| = 1$  and for all  $i \in [k]$ ,  $|R_i \cap X| = 1$ ?

SRBPC is a variant of PERFECT CODE which is known to be W[1]-hard [14]. The W[1]-hardness proof of SRBPC is given by the following theorem. **Proofs of the results marked by an asterik are omitted due to space constraints.**

► **Theorem 12 (\*)**. SRBPC parameterized by the number of parts in  $\mathcal{R}$  is W[1]-hard.

Let  $(G, \mathcal{R} = R_1 \uplus R_2 \uplus \dots \uplus R_k, \mathcal{B})$  be an instance of SRBPC. We will assume that  $|\mathcal{B}| = dk$ , otherwise, the instance is a trivial NO instance of SRBPC. For technical reasons we assume that  $|\mathcal{B}| = \ell > 4k$  (and hence  $d > 4$ ). Such an assumption is valid because otherwise, the problem is FPT. Indeed, if  $|\mathcal{B}| = \ell \leq 4k$  then for every partition  $P_1, \dots, P_k$  of  $\mathcal{B}$  into  $k$  parts such that each part is non-empty, we first guess a permutation  $\pi$  on  $k$  elements and then for every  $i \in [k]$ , we check whether there exists a vertex  $r_{\pi(i)} \in R_{\pi(i)}$  that dominates exactly all the vertices in  $P_i$  (and none in other parts  $P_j$ ,  $j \neq i$ ). Clearly, all this can be done in time  $2^{\mathcal{O}(k \log k)} n^{\mathcal{O}(1)}$ . Furthermore, we also assume that  $k \geq 2$ , else the problem is solvable in polynomial time. Now we give the desired reduction. We construct an instance  $(G', k')$  of SPLIT CONTRACTION as follows. Initially,  $V(G') = \mathcal{R} \cup \mathcal{B}$  and  $E(G') = E(G)$ . For all  $b, b' \in \mathcal{B}$ ,  $b \neq b'$ , we add the edge  $(b, b')$  to  $E(G')$ . That is, we transform  $\mathcal{B}$  into a clique. Let  $t = 2k + 2$ . For each  $b_i \in \mathcal{B}$ , we add a set of  $t$  vertices  $y_{i,1}^i, \dots, y_{i,t}^i$  each adjacent to  $b_i$  in  $G'$ . We add a vertex  $s$  adjacent to every vertex  $r \in \mathcal{R}$  in  $G'$ . Also, we add a set of  $t$  vertices  $q_1, \dots, q_t$  each adjacent to  $s$  in  $G'$ . For each  $i \in [k]$ , we add a vertex  $x_i$  adjacent to each vertex  $r \in R_i$ . Finally, for all  $i \in [k]$ , we add a set of  $t$  vertices  $w_{i,1}^i, \dots, w_{i,t}^i$  adjacent to  $x_i$  in  $G'$ . We set the new parameter  $k'$  to be  $2k$ . This completes the description of the reduction. We refer the reader to Figure 2 for an illustration of the reduction.

In the next four lemmata (Lemmata 13 to 16) we prove certain structural properties of the instance  $(G', k')$  of SPLIT CONTRACTION. These will later be used in showing that

$(G, \mathcal{R} = R_1 \uplus R_2 \uplus \dots \uplus R_k, \mathcal{B})$  is a YES instance of SRBPC if and only if  $(G', k')$  is a YES instance of SPLIT CONTRACTION. For the next four lemmata, we let  $S$  be a solution to SPLIT CONTRACTION in  $(G', k')$  and  $H = G'/S$  with  $\hat{C}, \hat{I}$  being a partition of  $V(H)$  inducing a clique and an independent set, respectively, in  $H$ . Let  $\varphi : V(G) \rightarrow V(H)$  denote the function defining the contractibility of  $G$  to  $H$ , and  $\mathcal{W}$  be the  $H$ -witness structure of  $G$ .

► **Lemma 13 (\*)**. *Let  $(G', k')$  be a YES instance of SPLIT CONTRACTION. Then, for all  $v \in (\{s\} \cup \mathcal{B} \cup \{x_i \mid i \in [k]\})$ , we have  $\varphi(v) \in \hat{C}$ .*

► **Lemma 14 (\*)**. *Let  $(G', k')$  be a YES instance of SPLIT CONTRACTION. Then, for all  $i \in [k]$ , there exists  $r_i \in R_i$  such that  $(x_i, r_i) \in S$ .*

For each  $i \in [k]$  we arbitrarily choose a vertex  $r_i^* \in R_i$  such that  $e_i^* = (x_i, r_i^*) \in S$ . The existence of such a vertex is guaranteed by Lemma 14.

► **Lemma 15 (\*)**. *Let  $(G', k')$  be a YES instance of SPLIT CONTRACTION. Then, for all  $i \in [k]$  and  $h_i = \varphi(r_i^*)$ , we have  $|W(h_i)| \geq 3$ . Furthermore, there is an edge  $e_i \neq e_i^*$  in  $S$  incident to exactly one of  $x_i, r_i^*$  and not incident to the vertices in  $\{w_1^i, \dots, w_t^i\}$ .*

From Lemma 14 we know that for each  $i \in [k]$ , we have  $r_i^* \in R_i$  such that  $(x_i, r_i^*) \in S$ . Similarly, from Lemma 15 we know that, for each  $i \in [k]$ , there is an edge incident to one of  $x_i, r_i^*$  other than  $e_i^* = (x_i, r_i^*)$  in every solution. Recall that for  $i, j \in [k]$ ,  $i \neq j$  none of  $x_i, r_i^*$  is adjacent to  $x_j, r_j^*$ . Hence, it follows that we have already used up our budget of  $k' = 2k$  by forcing certain types of edges to be in  $S$ . Finally, we prove Lemma 16 that forces even more structure on the witness sets.

► **Lemma 16**. *Let  $(G', k')$  be a YES instance of SPLIT CONTRACTION. Then, for all  $i \in [k]$ ,  $r_i^* \in W(\varphi(s))$ .*

**Proof.** Let  $h_s = \varphi(s)$  and  $\hat{R} = \{r_i^* \mid i \in [k], r_i^* \in W(h_s)\}$ . For a contradiction assume that  $|\hat{R}| < k$ , otherwise the claim trivially holds. By Lemma 14, for each  $i \in [k]$ ,  $e_i^* = (x_i, r_i^*) \in S$ . This implies that for all  $r_i^* \in \hat{R}$ ,  $x_i \in W(h_s)$  and hence  $|W(h_s)| \geq 2|\hat{R}| + 1$ . From Lemma 15 we know that there exists an edge  $e_i \neq e_i^* \in S$  incident to either  $x_i$  or  $r_i^*$  and not incident to any vertex in  $\{w_1^i, \dots, w_t^i\}$ . Thus, every edge in  $S$  is incident to either  $x_i$  or  $r_i^*$ . This implies that for every vertex  $z \in \{q_1, \dots, q_t\} \cup \{y_1^j, \dots, y_t^j \mid j \in [\ell]\}$ ,  $|W(\varphi(z))| = 1$ . Now we show that there exists a vertex in  $\mathcal{B}$  that is not adjacent to any vertex in  $W(h_s)$ . Observe that the only vertices in  $W(h_s)$  that can be adjacent to a vertex in  $\mathcal{B}$  are in  $\hat{R}$ . However, every vertex in  $\hat{R}$  has exactly  $d$  neighbours in  $\mathcal{B}$ . This together with the fact that  $|\mathcal{B}| = \ell = dk > d|\hat{R}|$  implies that there exists a subset  $\mathcal{B}'$  of size  $d(k - |\hat{R}|)$  such that none of these vertices are adjacent to any vertex in  $\hat{R}$ . However, at most  $(k - |\hat{R}|)$  vertices in  $\mathcal{B}'$  can be incident to an edge in  $S$ . This implies that there exists a vertex  $b \in \mathcal{B}'$  with  $h = \varphi(b)$  such that it is not incident to any edge in  $S$  and thus  $|W(h)| = 1$ . But then we can conclude that  $W(h)$  and  $W(h_s)$  are not adjacent in  $G'$ . However, by Lemma 13 we know that  $h_s, h \in \hat{C}$  and thus there is an edge  $(h = \varphi(b), h_s) \in E(H')$ , a contradiction. This contradicts our assumption that  $|\hat{R}| < k$  and proves the claim. ◀

We are now ready to prove the equivalence between the instance  $(G, \mathcal{R}, \mathcal{B})$  of SRBPC and the instance  $(G', k')$  of SPLIT CONTRACTION.

► **Lemma 17**.  *$(G, \mathcal{R} = R_1 \uplus \dots \uplus R_k, \mathcal{B})$  is a YES instance of SRBPC if and only if  $(G', k')$  is a YES instance of SPLIT CONTRACTION.*

**Proof.** In the forward direction, let  $Z = \{r_i \mid r_i \in R_i, i \in [k]\} \subseteq \mathcal{R}$  be a solution to  $(G, \mathcal{R}, \mathcal{B})$  of SRBPC. Let  $Z' = \{(r_i, x_i), (r_i, s) \mid i \in [k]\}$ . Observe that  $|Z'| = 2k$ . Let  $T = \{r_i, x_i \mid i \in [k], r_i \in Z\}$ . We define the following surjective function  $\varphi : V(G') \rightarrow V(G') \setminus T$ . If  $v \in T \cup \{s\}$  then  $\varphi(v) = s$ , else  $\varphi(v) = v$ . Observe that  $G'[W(s)]$  is connected and for all  $v \in V(G') \setminus (T \cup \{s\})$ ,  $W(v)$  is a singleton set. Consider the graph  $H$  with  $V(H) = V(G') \setminus T$  and  $(v, u) \in E(H)$  if and only if  $\varphi^{-1}(v), \varphi^{-1}(u)$  are adjacent in  $G'$ . Note that the graphs  $G'/Z'$  and  $H$  are isomorphic, therefore we prove that  $H$  is a split graph. Let  $\hat{C} = \{\varphi(v) \mid v \in \mathcal{B} \cup \{s\}\}$  and  $\hat{I} = V(H) \setminus \hat{C}$ . For  $v, u \in \hat{I}$ ,  $\varphi^{-1}(v) = \{v\}$  and  $\varphi^{-1}(u) = \{u\}$  and  $\{v\}, \{u\}$  are non-adjacent in  $G'$ . Therefore,  $(v, u) \notin E(H)$ . This proves that  $\hat{I}$  is an independent set in  $H$ . For  $b, b' \in \mathcal{B} \subset \hat{C}$ ,  $(b, b') \in E(G')$ , therefore  $(\varphi(v), \varphi(u)) \in E(H)$ . Since  $Z$  is a solution to SRBPC in  $(G, \mathcal{R}, \mathcal{B})$ , for each  $b \in \mathcal{B}$ , there exists  $r_i \in Z$  such that  $(b, r_i) \in E(G')$ , therefore,  $W(s)$  and  $b$  are adjacent in  $G'$ . Hence,  $(\varphi(s), \varphi(b)) \in E(H)$ . This finishes the proof that  $\hat{C}$  induces a clique in  $H$  and that  $H$  is a split graph.

In the reverse direction, let  $S$  be a solution to  $(G', k')$  of SPLIT CONTRACTION, and denote  $H = G'/S$ . Let  $W$  be the  $H$ -witness structure of  $G'$ ,  $\varphi$  be the associated surjective function and  $h_s = \varphi(s)$ . From Lemmas 14 and 16 it follows that for all  $i \in [k]$ , there exists  $r_i^* \in R_i$  such that  $(x_i, r_i^*) \in S$  and  $r_i^*, x_i \in W(h_s)$ . Let  $Z = \{r_i^* \mid i \in [k]\}$ . We will show that  $Z$  is a solution to SRBPC in  $(G, \mathcal{R}, \mathcal{B})$ . Since  $|W(h_s)| = k' + 1 = 2k + 1$ , it holds that for all  $v \in V(H) \setminus \{h_s\}$ ,  $|W(v)| = 1$ . This implies that for all  $b \in \mathcal{B}$ ,  $b \notin W(h_s)$ . Also observe that since  $x_i \in W(h_s)$  for all  $i \in [k]$  and  $|W(h_s)| = k' + 1 = 2k + 1$ , we have that  $|W(h_s) \cap R_i| = 1$ . This implies that  $|Z| = k$  and  $|Z \cap R_i| = 1$ , for all  $i \in [k]$ . To show that  $Z$  is indeed a solution, it is enough to show that for all  $b_j \in \mathcal{B}$ ,  $|Z \cap N(b_j)| = 1$ . Towards a contradiction, assume there exists  $b_j \in \mathcal{B}$  such that  $|Z \cap N(b_j)| \neq 1$ . Let  $h_{b_j} = \varphi(b_j)$ . We consider the following two cases.

- If  $|Z \cap N(b_j)| < 1$ . Recall that  $W(h_{b_j}) = \{b_j\}$ . Further,  $N_{G'}(b_j) \subseteq \mathcal{R} \cup \{y_1^j, \dots, y_t^j\}$ ,  $Z = W(h_s) \cap \mathcal{R}$  and by our assumption  $Z \cap N_{G'}(b_j) = \emptyset$ . But then  $W(h_s)$  and  $W(h_{b_j})$  are not adjacent in  $G'$ . However, Lemma 13 implies that  $(h_s, h_{b_j}) \in E(H)$ , contradicting our assumption that  $|Z \cap N(b_j)| < 1$ .
- If  $|Z \cap N(b_j)| > 1$ , then there exists  $j, j' \in [k]$ ,  $j \neq j'$  such that  $r_j^*, r_{j'}^* \in N_{G'}(b)$ . Then it follows that  $|\cup_{i \in [k]} N(r_i^*)| < \ell = dk$ . But then there exists  $b' \in \mathcal{B}$  such that  $W(\varphi(b'))$  and  $W(h_s)$  are non-adjacent, contradicting that  $(\varphi(b'), h_s) \in E(H)$  from Lemma 13.

This completes the proof. ◀

We now restate Theorem 2.

► **Theorem 2 (restated).** SPLIT CONTRACTION parameterized by the size of a solution is  $W[1]$ -hard.

**Proof.** Proof follows from construction, Lemma 17 and Theorem 12. ◀

## 5 Conclusion

In this paper, we have established two important results regarding the complexity of SPLIT CONTRACTION. First, we have shown that under the ETH, this problem cannot be solved in time  $2^{o(\ell^2)} \cdot n^{O(1)}$  where  $\ell$  is the vertex cover number of the input graph, and this lower bound is tight. To the best of our knowledge, this is the first tight lower bound of the form  $2^{o(\ell^2)} \cdot n^{O(1)}$  for problems parameterized by the vertex cover number of the input graph. Second, we have proved that SPLIT CONTRACTION, despite its deceptive simplicity, is actually  $W[1]$ -hard with respect to the solution size. We believe that techniques integrated in our



constructions can be used to derive conditional lower bounds and  $W[1]$ -hardness results in the context of other graph editing problems where the edit operation is edge contraction.

We would like to conclude our paper with the following intriguing question. In the exact setting, it is easy to see that SPLIT CONTRACTION can be solved in time  $2^{\mathcal{O}(n \log n)}$ . Can it be solved in time  $2^{o(n \log n)}$ ? A negative answer would imply, for instance, that it is neither possible to find a topological clique minor in a given graph in time  $2^{o(n \log n)}$ , which is an interesting open problem [11]. It might be possible that tools developed in our paper, such as the usage of harmonious coloring, can be utilized to shed light on such problems.

---

## References

- 1 Takao Asano and Tomio Hirata. Edge-contraction problems. *Journal of Computer and System Sciences*, 26(2):197–208, 1983.
- 2 Ivan Bliznets, Marek Cygan, Pawel Komosa, Lukas Mach, and Michal Pilipczuk. Lower bounds for the parameterized complexity of minimum fill-in and other completion problems. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1132–1151, 2016.
- 3 Ivan Bliznets, Fedor V. Fomin, Marcin Pilipczuk, and Michal Pilipczuk. A subexponential parameterized algorithm for proper interval completion. In *European Symposium on Algorithms (ESA)*, pages 173–184, 2014.
- 4 Ivan Bliznets, Fedor V. Fomin, Marcin Pilipczuk, and Michal Pilipczuk. A subexponential parameterized algorithm for interval completion. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1116–1131, 2016.
- 5 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.
- 6 Leizhen Cai and Chengwei Guo. Contracting few edges to remove forbidden induced subgraphs. In *International Workshop on Parameterized and Exact Computation (IPEC)*, pages 97–109, 2013.
- 7 Yixin Cao. Unit interval editing is fixed-parameter tractable. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 306–317, 2015.
- 8 Yixin Cao. Linear recognition of almost interval graphs. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1096–1115, 2016.
- 9 Yixin Cao and Dániel Marx. Chordal editing is fixed-parameter tractable. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 214–225, 2014.
- 10 Yixin Cao and Dániel Marx. Interval deletion is fixed-parameter tractable. *ACM Transactions on Algorithms (TALG)*, 11(3):21, 2015.
- 11 Marek Cygan, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socala. Tight bounds for graph homomorphism and subgraph isomorphism. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1643–1649, 2016.
- 12 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 13 Marek Cygan, Marcin Pilipczuk, and Michal Pilipczuk. Known algorithms for edge clique cover are probably optimal. *SIAM Journal on Computing*, 45(1):67–83, 2016.
- 14 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for  $W[1]$ . *Theoretical Computer Science*, 141(1&2):109–131, 1995.
- 15 Pål Grønås Drange and Michal Pilipczuk. A polynomial kernel for trivially perfect editing. In *European Symposium on Algorithms (ESA)*, pages 424–436, 2015.
- 16 Pål Grønås Drange, Markus S. Dregi, Daniel Lokshtanov, and Blair D. Sullivan. On the threshold of intractability. In *European Symposium on Algorithms (ESA)*, pages 411–423, 2015.

- 17 Pål Grønås Drange, Fedor V. Fomin, Michał Pilipczuk, and Yngve Villanger. Exploring subexponential parameterized complexity of completion problems. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 288–299, 2014.
- 18 Keith Edwards. The harmonious chromatic number and the achromatic number. *Surveys in Combinatorics*, pages 13–48, 1997.
- 19 Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Yngve Villanger. Tight bounds for parameterized complexity of cluster editing with a small number of clusters. *Journal of Computer and System Sciences*, 80(7):1430–1447, 2014.
- 20 Fedor V. Fomin and Yngve Villanger. Subexponential parameterized algorithm for minimum fill-in. *SIAM Journal on Computing*, 42(6):2197–2216, 2013.
- 21 Esha Ghosh, Sudeshna Kolay, Mrinal Kumar, Pranabendu Misra, Fahad Panolan, Ashutosh Rai, and M. S. Ramanujan. Faster parameterized algorithms for deletion to split graphs. *Algorithmica*, 71(4):989–1006, 2015.
- 22 Petr A. Golovach, Pim van ’t Hof, and Daniel Paulusma. Obtaining planarity by contracting few edges. *Theoretical Computer Science*, 476:38–46, 2013.
- 23 Sylvain Guillemot and Dániel Marx. A faster FPT algorithm for bipartite contraction. *Information Processing Letters*, 113(22–24):906–912, 2013.
- 24 Chengwei Guo and Leizhen Cai. Obtaining split graphs by edge contraction. *Theoretical Computer Science*, 607:60–67, 2015.
- 25 Pinar Heggeres, Pim van ’t Hof, Daniel Lokshtanov, and Christophe Paul. Obtaining a bipartite graph by contracting few edges. *SIAM Journal on Discrete Mathematics*, 27(4):2143–2156, 2013.
- 26 Pinar Heggeres, Pim van ’t Hof, Benjamin Lévêque, Daniel Lokshtanov, and Christophe Paul. Contracting graphs to paths and trees. *Algorithmica*, 68(1):109–132, 2014.
- 27 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.
- 28 Christian Komusiewicz. Tight running time lower bounds for vertex deletion problems. *arXiv preprint arXiv:1511.05449*, 2015.
- 29 Sin-Min Lee and John Mitchem. An upper bound for the harmonious chromatic number. *Journal of Graph Theory*, 11(4):565–567, 1987.
- 30 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 760–776, 2011.
- 31 Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh. On the hardness of eliminating small induced subgraphs by contracting edges. In *International Workshop on Parameterized and Exact Computation (IPEC)*, pages 243–254, 2013.
- 32 Dániel Marx and Valia Mitsou. Double-exponential and triple-exponential bounds for choosability problems parameterized by treewidth. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 28:1–28:15, 2016.
- 33 Colin McDiarmid and Luo Xinhua. Upper bounds for harmonious coloring. *Journal of Graph Theory*, 15(6):629–636, 1991.
- 34 Toshimasa Watanabe, Tadashi Ae, and Akira Nakamura. On the removal of forbidden graphs by edge-deletion or by edge-contraction. *Discrete Applied Mathematics*, 3(2):151–153, 1981.
- 35 Toshimasa Watanabe, Tadashi Ae, and Akira Nakamura. On the NP-hardness of edge-deletion and-contraction problems. *Discrete Applied Mathematics*, 6(1):63–78, 1983.

# The Operator Approach to Entropy Games\*

Marianne Akian<sup>1</sup>, Stéphane Gaubert<sup>2</sup>, Julien Grand-Clément<sup>3</sup>, and Jérémie Guillaud<sup>4</sup>

- 1 INRIA and CMAP, École polytechnique, CNRS, Palaiseau, France  
marianne.akian@inria.fr
- 2 INRIA and CMAP, École polytechnique, CNRS, Palaiseau, France  
stephane.gaubert@inria.fr
- 3 École polytechnique, Palaiseau, France  
julien.grand-clement@polytechnique.edu
- 4 École polytechnique, Palaiseau, France  
jeremie.guillaud@polytechnique.edu

---

## Abstract

Entropy games and matrix multiplication games have been recently introduced by Asarin et al. They model the situation in which one player (Despot) wishes to minimize the growth rate of a matrix product, whereas the other player (Tribune) wishes to maximize it. We develop an operator approach to entropy games. This allows us to show that entropy games can be cast as stochastic mean payoff games in which some action spaces are simplices and payments are given by a relative entropy (Kullback-Leibler divergence). In this way, we show that entropy games with a fixed number of states belonging to Despot can be solved in polynomial time. This approach also allows us to solve these games by a policy iteration algorithm, which we compare with the spectral simplex algorithm developed by Protasov.

**1998 ACM Subject Classification** G.2.1 Combinatorial Algorithms, F.2.1 Numerical Algorithms and Problems

**Keywords and phrases** Stochastic games, Shapley operators, policy iteration, Perron eigenvalues, Risk sensitive control

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.6

## 1 Introduction

### 1.1 Entropy games and matrix multiplication games

Entropy games have been introduced by Asarin et al. [5]. They model the situation in which two players with conflicting interests, called “Despot” and “Tribune”, wish to minimize or to maximize a topological entropy representing the freedom of a half-player, “People”. Entropy games are special “matrix multiplication games”, in which two players alternatively choose matrices in certain prescribed sets; the first player wishes to minimize the growth rate of the infinite matrix product obtained in this way, whereas the second player wishes to maximize it. Whereas general matrix multiplication games are hard in general (computing joint spectral radii is a special case), entropy games correspond to a tractable subclass of multiplication games, in which the matrix sets have the property of being invariant by row interchange, the so called independent row uncertainty (IRU) assumption. In particular, Asarin et al. showed

---

\* The authors were partially supported by the ANR through the MALTHY INS project, and by the PGMO program of FMJH and EDF.



in [5] that the problem of comparing the value of an entropy game to a given rational number is in  $\text{NP} \cap \text{coNP}$ , giving to entropy games a status somehow comparable to other important classes of games with an unsettled complexity, including mean payoff games, simple stochastic games, or stochastic mean payoff games, see [4] for background.

Another motivation to study entropy games arises from risk sensitive control [13, 14, 3]: as we shall see, essentially the same class of operators arise in the latter setting. Further motivations originate from symbolic dynamics [21, Chapter 1.8.4].

## 1.2 Contribution

We first show that entropy games, which were introduced as a new class of games, are equivalent to a class of zero-sum mean payoff stochastic games with perfect information, in which some action spaces are simplices, and the instantaneous payments are given by a Kullback-Leibler entropy. Hence, entropy games fit in a classical class of games, with a “nice” payment function over infinite action spaces.

To do so, we introduce a more expressive variant of the model of Asarin et al [5], called here *extended* entropy games for clarity, in which the initial state is prescribed (the initial state is chosen by one player, People, in the original model). This extension is needed to develop an operator approach and derive consequences from it. We show that the main results known for stochastic mean payoff games with finite actions space, namely the existence of the value and the existence of optimal positional strategies, are still valid for extended entropy games (Theorems 2 and 3). This is derived from a model theory approach of Bolte, Gaubert, and Vigeral [8], together with the observation that the dynamic programming operators of extended entropy games are definable in the real exponential field. Another consequence of the operator approach is the existence of Collatz-Wielandt optimality certificates for entropy games, Theorem 12. When specialized to the one player case, this leads to a convex programming characterization of the value, Corollary 13, which can also be recovered from a characterization of Anantharam and Borkar [3].

This leads us to our main result, Theorem 14, showing that (extended) entropy games in which Despot has a fixed number of significant states (states with a nontrivial choice) can be solved in polynomial time. Thus, entropy games are somehow similar to stochastic mean payoff games, for which an analogous fixed-parameter tractability result holds (by reducing the one player case to a linear program). This also reveals a fundamental asymmetry between the players Despot and Tribune: our approach does not lead to a polynomial bound if one fixes the number of states of Tribune. The proof relies on several ingredients: ellipsoid method, separation bounds between algebraic numbers, results from Perron-Frobenius theory.

The operator approach also allows one to obtain practically efficient algorithms to solve entropy games. In this way, the classical policy iteration of Hoffman-Karp [19] can be adapted to entropy games. We report experiments showing that when specialized to one player problems, policy iteration yields a speedup by one order of magnitude by comparison with the “spectral simplex” method recently introduced by Protasov [23].

Let us finally complete the discussion of related works. The formulation of entropy games in terms of “classical” mean payoff games in which the payments are given by a Kullback-Leibler entropy builds on known principles in risk sensitive control [14, 3]. It can be thought as a version for two player problems of the Donsker-Varadhan characterization of the Perron-eigenvalue [11]. A Donsker-Varadhan type formula for risk sensitive problems, which can be applied in particular to Despot-free player entropy games, has been recently obtained by Anantharam and Borkar, in a wider setting allowing an infinite state space [3]. In a nutshell, for Despot-free problems, the Donsker-Varadhan formula appears to be the

(convex-analytic) dual of the Collatz-Wielandt formula. Chen and Han [10] developed a related convex programming approach to solve the entropy maximization problem for Markov chains with uncertain parameters. We also note that the present Collatz-Wielandt approach, building on [2], yields an alternative to the approach of [5] using the “hourglass alternative” of [20] to produce concise certificates allowing one to bound the value of entropy games. Finally, the identification of tractable subclasses of matrix multiplication games can be traced back at least to the work of Blondel and Nesterov [7].

## 2 Entropy games

### 2.1 Entropy games with prescribed initial state

An extended entropy game  $\Gamma^{\text{ent}}$  is a perfect information game played on a finite directed weighted graph  $G$ . There are 2 players, “Despot”, “Tribune”, and a half-player with a nondeterministic behavior, “People”. The set of nodes of the graph is written as the disjoint union  $D \cup T \cup P$ , where  $D, T$  and  $P$  represent sets of states in which Despot, Tribune, and People play. We assume that the set of arcs  $E$  is included in  $(D \times T) \cup (T \times P) \cup (P \times D)$ , meaning that Despot, Tribune, and People alternate their actions. A *weight*  $m_{pd}$ , which is a positive real number, is attached to every arc  $(p, d) \in P \times D$ . All the other arcs in  $E$  have weight 1. An initial state,  $\bar{d} \in D$ , is known to the players. A token, initially in node  $\bar{d}$ , is moved in the graph according to the following rule. If the token is currently in a node  $d$  belonging to  $D$ , then, Despot chooses an arc  $(d, t) \in E$  and moves the token to node  $t$ . Similarly, if the token is currently in a node  $t \in T$ , Tribune chooses an arc  $(t, p) \in E$  and moves the token to node  $p$ . Finally, if the token is in a node  $p \in P$ , People chooses an arc  $(p, d') \in E$  and moves the token to a node  $d' \in D$ . We will assume that every player has at least one possible action in each state in which it is his or her turn to play. In other words, for all  $d \in D$ , the set of actions  $\{(d, t) \in E\}$  must be nonempty, and similar conditions apply to  $t \in T$  and  $p \in P$ .

A *history* of the game consists of a finite path in the digraph  $G$ , starting from the initial node  $\bar{d}$ . The *number of turns* of this history is defined to be the length of this path, each arc counting for a length of one third. The *weight* of a history is defined to be the product of the weights of the arcs arising on this path. For instance, a history  $(d_0, t_0, p_0, d_1, t_1, p_1, d_2, t_2)$  where  $d_i \in D$ ,  $t_i \in T$  and  $p_i \in P$ , makes 2 and 1/3 turn, and its weight is  $m_{p_0 d_1} m_{p_1 d_2}$ .

A *strategy* of Player Despot is a map  $\delta$  which assigns to every history ending in some node  $d$  in  $D$  an arc of the form  $(d, t) \in E$ . Similarly, a *strategy* of Player Tribune is a map  $\tau$  which assigns an arc  $(t, p) \in E$  to every history ending with a node  $t$  in  $T$ .

For every integer  $k$ , we define as follows the *game in horizon  $k$*  with initial state  $\bar{d}$ ,  $\Gamma^{\text{ent}}(k, \bar{d})$ . We assume that Despot and Tribune play according to the strategies  $\delta, \tau$ . Then, People plays in a nondeterministic way. Therefore, the pair of strategies  $\delta, \tau$  allows for different histories. The payment received by Tribune, in  $k$  turns, is denoted by  $R_{\bar{d}}^k(\delta, \tau)$ . It is defined as the sum of the weights of all the paths of the digraph  $G$  of length  $k$  with initial node  $\bar{d}$  determined by the strategies  $\delta$  and  $\tau$ : each of these paths corresponds to different successive choices of People, leading to different histories allowed by the strategies  $\delta, \tau$ . The payment received by Despot is defined to be the opposite of  $R_{\bar{d}}^k(\delta, \tau)$ , so that the game in horizon  $k$  is zero-sum. In that way, the payment  $R_{\bar{d}}^k$  measures the “freedom” of People, Despot wishes to minimize it whereas Tribune wishes to maximize it.

We say that the game  $\Gamma^{\text{ent}}(k, \bar{d})$  in horizon  $k$  with initial state  $\bar{d}$  has the value  $V_{\bar{d}}^k$  and that  $\delta^*, \tau^*$  are *optimal strategies* of Despot and Tribune if for all strategies  $\delta, \tau$  of Despot

## 6:4 The Operator Approach to Entropy Games

and Tribune, we have the saddle point property:

$$R_{\bar{d}}^k(\delta, \tau^*) \geq R_{\bar{d}}^k(\delta^*, \tau^*) = V_{\bar{d}}^k \geq R_{\bar{d}}^k(\delta^*, \tau) . \quad (1)$$

If the value  $V_{\bar{d}}^k$  exists for all choices of the initial state  $\bar{d}$ , we define the *value vector* of the game  $\Gamma^{\text{ent}}(k, \cdot)$  in horizon  $k$ , to be  $V^k := (V_{\bar{d}}^k)_{\bar{d} \in D} \in \mathbb{R}^D$ .

We now define the *infinite horizon game*  $\Gamma^{\text{ent}}(\infty, \bar{d})$ , in which the payment received by Tribune is given by

$$R_{\bar{d}}^{\infty}(\delta, \tau) := \limsup_{k \rightarrow \infty} (R_{\bar{d}}^k(\delta, \tau))^{1/k}$$

and the payment received by Despot is the opposite of the latter payment. (The choice of limsup is somehow arbitrary, we could choose liminf instead without affecting the results which follow.) The *value*  $V_{\bar{d}}^{\infty}$  of the infinite horizon game  $\Gamma^{\text{ent}}(\infty, \bar{d})$ , and the optimal strategies in this game, are still defined by a saddle point condition, as in (1), the payment  $R_{\bar{d}}^k(\delta, \tau)$  being now replaced by  $R_{\bar{d}}^{\infty}(\delta, \tau)$ .

We denote by  $V^{\infty} = (V_{\bar{d}}^{\infty})_{\bar{d} \in D} \in \mathbb{R}^D$  the *value vector* of the infinite energy game  $\Gamma^{\text{ent}}(\infty, \cdot)$ .

We associate to the latter games the dynamic programming operator  $F : \mathbb{R}^D \rightarrow \mathbb{R}^D$ , such that, for all  $X \in \mathbb{R}^D$ , and  $d \in D$ ,

$$F_d(X) = \min_{(d,t) \in E} \max_{(t,p) \in E} \sum_{(p,d') \in E} m_{pd'} X_{d'} . \quad (2)$$

The existence of the value for the *finite* horizon game follows from a standard dynamic programming argument.

► **Proposition 1.** *The value of the extended entropy game in horizon  $k$ ,  $\Gamma^{\text{ent}}(k, \cdot)$ , does exist. The value vector  $V^k$  of this game is determined by the relations  $V^0 = e$ ,  $V^k = F(V^{k-1})$ ,  $k = 1, 2, \dots$ , where  $e$  is the unit vector of  $\mathbb{R}^D$ .*

Recall that a strategy is said to be *positional* or is called a *policy* if the decision taken at a given stage depends only on the last state which has been visited. The following theorem follows from Theorem 9 stated in Section 3, by using an equivalence with a special class of stochastic mean payoff games with infinite actions spaces, through log-glasses.

► **Theorem 2.** *The infinite horizon extended entropy game has a value and it has optimal positional strategies. Moreover, for all initial states  $d$ ,  $V_d^{\infty} = \lim_{k \rightarrow \infty} (V_d^k)^{1/k}$ .*

The following result is deduced from Theorem 11, using the same technique as for Theorem 2. We denote by  $\mathcal{P}_D$  (resp.  $\mathcal{P}_T$ ) the set of policies (i.e., positional strategies) of Despot (resp. Tribune). If one fixes a strategy  $\delta \in \mathcal{P}_D$  or  $\tau \in \mathcal{P}_T$ , we end up with a one player infinite horizon entropy game (a two player game in which either Despot or Tribune has no options), whose value is denoted by  $V_d^{\infty}(\delta, \star)$  (resp.  $V_d^{\infty}(\star, \tau)$ ). Similarly, if we fix the two strategies, we end up in a game in which only People has options, and the value of this game, denoted by  $V_d^{\infty}(\delta, \tau)$ , coincides with  $R_{\bar{d}}^{\infty}(\delta, \tau)$ .

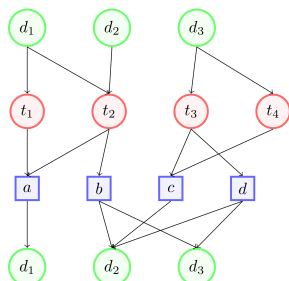
► **Theorem 3.** *We have*

$$V_d^{\infty} = \min_{\delta \in \mathcal{P}_D} V_d^{\infty}(\delta, \star) = \max_{\tau \in \mathcal{P}_T} V_d^{\infty}(\star, \tau) .$$

Moreover, for all  $\delta \in \mathcal{P}_D$  and for all  $\tau \in \mathcal{P}_T$ ,

$$V_d^{\infty}(\delta, \star) = \max_{\tau' \in \mathcal{P}_T} V_d^{\infty}(\delta, \tau'), \quad V_d^{\infty}(\star, \tau) = \min_{\delta' \in \mathcal{P}_D} V_d^{\infty}(\delta', \tau),$$

► **Example 4.** Take  $D = \{d_1, d_2, d_3\}$ ,  $T = \{t_1, t_2, t_3, t_4\}$ ,  $P = \{a, b, c, d\}$ ,  $E = \{(d_1, t_1), (d_1, t_2), (d_2, t_2), (d_3, t_3), (d_3, t_4), (t_1, a), (t_2, a), (t_2, b), (t_3, c), (t_3, d), (t_4, c), (a, d_1), (b, d_2), (b, d_3), (c, d_2), (d, d_2), (d, d_3)\}$  and  $m_{pd_i} = 1$  for all  $p \in P$  and  $1 \leq i \leq 3$  such that  $(p, d_i) \in E$ . The corresponding graph and dynamic programming operator are given by:



$$F_1(X) = \min(X_1, \max(X_1, X_2 + X_3)),$$

$$F_2(X) = \max(X_1, X_2 + X_3),$$

$$F_3(X) = \min(\max(X_2, X_2 + X_3), X_2).$$

One can check that  $V^k = (1, \phi_{k+1}, \phi_k)$ , where  $\phi_0 = \phi_1 = 1$  and  $\phi_{k+2} = \phi_k + \phi_{k+1}$  is the Fibonacci sequence. Hence, by Theorem 2, the value vector of this entropy game is  $V^\infty = (1, \omega, \omega)$  where  $\omega := (1 + \sqrt{5})/2$ .

## 2.2 The original entropy game model

The original entropy game model of Asarin et al. [5] is a zero-sum game defined in a similar way, up to a difference: in their model, the initial state is not prescribed. The payment of Tribune in horizon  $k$ , instead of being  $R_d^k(\delta, \tau)$ , is the quantity  $\bar{R}^k(\delta, \tau)$ , defined now as the sum of weights of all paths of length  $k$  starting at a node in  $D$  and ending at a node in  $D$ . Hence,  $\bar{R}^k(\delta, \tau) = \sum_{d \in D} R_d^k(\delta, \tau)$ . The payment of Tribune can be defined in their game as follows  $\bar{R}^\infty(\delta, \tau) = \limsup_{k \rightarrow \infty} (\bar{R}^k(\delta, \tau))^{1/k}$ . This game is denoted by  $\Gamma^{\text{ent}}(\infty)$ , we denote by  $\bar{V}^\infty$  the value of this game, which is shown to exist in [5].

Note that in the initial model in [5], the weights  $m_{pd'}$  are equal to 1. The generalization to weighted entropy games, in which the weights  $m_{pd'}$  are integers is discussed in Section 6 of [5]. The case in which the weights  $m_{pd'}$  take rational values can be reduced to the latter case by multiplying all the weights by an integer factor. Therefore, we will ignore the restriction that  $m_{pd'} = 1$  in our definition of  $\Gamma^{\text{ent}}(\infty)$  and will refer to the entropy game model with rational weights as the entropy game model. The next result, which can be deduced from the existence of the value of the extended entropy game (Theorem 2 above), shows that the value of the original entropy game can be recovered from the value vector of the extended one:

► **Proposition 5.** *The value of the original entropy game  $\Gamma^{\text{ent}}(\infty)$  coincides with the maximum of the values of the extended entropy games  $\Gamma^{\text{ent}}(\infty, d)$ , taken over all initial states:  $\bar{V}^\infty = \max_{d \in D} V_d^\infty$ .*

► **Example 6.** This is illustrated by the game of Example 4. In the original model of [5], the value, defined independently of the initial state, is  $(1 + \sqrt{5})/2$ , whereas our model associates to the initial state  $d_1$  a value 1 which differs from the values of  $d_2$  and  $d_3$ .

In [5], entropy games were compared with matrix multiplication games. We present here this correspondence in the case of general weights  $m_{pd'}$ . Given policies  $\delta \in \mathcal{P}_D$  and  $\tau \in \mathcal{P}_T$ , let  $A(\delta) \in \mathbb{R}^{D \times T}$  and  $B(\tau) \in \mathbb{R}^{T \times D}$  be such that  $A(\delta)_{dt} = 1$  if  $t = \delta(d)$  and 0 otherwise, and  $B(\tau)_{td} = m_{\tau(t)d}$  if  $(\tau(t), d) \in E$  and 0 otherwise, for all  $(d, t) \in D \times T$ . We shall think of  $A(\delta)$  and  $B(\tau)$  as rectangular matrices. Then  $\bar{R}^k(\delta, \tau) = \|(A(\delta)B(\tau))^k\|_1$ ,

## 6:6 The Operator Approach to Entropy Games

where for any  $A \in \mathbb{R}^{D \times D}$ ,  $A^k$  denotes its  $k$ th power and  $\|A\|_1 = \sum_{dd'} |A_{dd'}|$  its  $\ell^1$  norm. From this, one deduces that  $\bar{R}^\infty(\delta, \tau) = \rho(A(\delta)B(\tau))$ , where  $\rho(A)$  denotes the spectral radius of the matrix  $A$ . Moreover, let  $\mathcal{A}$  and  $\mathcal{B}$  denote the sets of all matrices of the form  $A(\delta)$  and  $B(\tau)$  respectively, and let  $\mathcal{AB}$  be the set of all matrices  $AB$  with  $A \in \mathcal{A}$  and  $B \in \mathcal{B}$ . The sets  $\mathcal{A}$ ,  $\mathcal{B}$  and  $\mathcal{AB}$  are subsets of matrices  $\mathcal{M}$  satisfying the property that all elements of  $\mathcal{M}$  have same dimension and if  $\mathcal{M}_i$  is the set of  $i$ th rows of the elements of  $\mathcal{M}$ , then  $\mathcal{M}$  is the set of matrices the  $i$ th row of which belongs to  $\mathcal{M}_i$ . Such a property defines the notion of IRU matrix sets (for independent row uncertainty sets) in [5]. The following property proved in [5] is the analogue of Theorem 3,  $V_d^\infty$  being replaced by  $\bar{V}^\infty$ :

$$\bar{V}^\infty = \min_{A \in \mathcal{A}} \max_{B \in \mathcal{B}} \rho(AB) = \max_{B \in \mathcal{B}} \min_{A \in \mathcal{A}} \rho(AB) . \quad (3)$$

A more general property is proved in [2, Section 8], as a consequence of the Collatz-Wielandt theorem (see Theorem 12 below).

### 3 Stochastic mean payoff game with Kullback-Leibler payments

We next show that extended entropy games are equivalent to a class of mean payoff games in which some action spaces are simplices, and payments are given by the Kullback-Leibler divergence.

To the extended entropy games  $\Gamma^{\text{eent}}$ , we associate a family of stochastic zero-sum games with Kullback-Leibler payments, denoted  $\Gamma^{\text{kl}}$  and defined as follows. These new games are still played on the weighted digraph  $G$ . For any node  $p \in P$ , we denote by  $E_p := \{(p, d) \in E\}$  the set of actions available to People in state  $p$ , and we denote by  $\Delta_p$  the set of probability measures on  $E_p$ . Therefore, an element of  $\Delta_p$  can be identified to a vector  $\nu = (\nu_{p,d})_{(p,d) \in E_p}$  with nonnegative entries and sum 1. The actions of Despot and Tribune in the states  $d \in D$  and  $t \in T$  are the same in the games  $\Gamma^{\text{kl}}$  and in the games  $\Gamma^{\text{eent}}$ . However, the two games have different rules when the state is in  $P$ , since the nondeterministic half-player, People, is now replaced by a standard probabilistic half-player, Nature. In the game  $\Gamma^{\text{kl}}$ , Tribune, who arrived in a state  $p \in P$  by choosing first an action in some state  $t \in T$ , so that  $(t, p) \in E$ , has to play again in state  $p$ , by choosing a probability measure  $\nu \in \Delta_p$ . Then, Nature chooses the next state  $d$  according to probability  $\nu_{p,d}$ , and Tribune receives the payment  $-S_p(\nu; m)$ , where  $S_p(\nu; m)$  is the relative entropy or Kullback-Leibler divergence:

$$S_p(\nu; m) := \sum_{(p,d) \in E_p} \nu_{pd} \log(\nu_{pd}/m_{pd}) .$$

An interesting special case arises when  $m \equiv 1$ , as in [5]. Then,  $S_p(\nu; m) = S_p(\nu) := \sum_{(p,d) \in E_p} \nu_{pd} \log \nu_{pd}$  is nothing but the Shannon entropy of  $\nu$ .

A history in the game  $\Gamma^{\text{kl}}$  now consists of a finite sequence  $(d_0, t_0, p_0, \nu_0, d_1, t_1, p_1, \dots)$ , which encodes both the states and actions which have been chosen. A strategy  $\delta$  of Despot is still a function which associates to a history ending in a state in  $d$  an arc  $(d, t)$  in  $E_d := \{(d, t) \in E\}$ . A strategy of Tribune has now two components  $(\tau, \pi)$ ,  $\tau$  is a map which assigns to a history ending in a state in  $t$  an arc  $(t, p) \in E$ , as before, whereas  $\pi$  assigns to the same history and to the next state  $p = \tau(d)$  chosen according to  $\tau$  a probability measure on  $\Delta_p$ . To each history corresponds a path in  $G$ , obtained by ignoring the occurrences of probability measures. For instance, the path corresponding to the history  $h = (d_0, t_0, p_0, \nu_0, d_1, t_1, p_1)$  is  $(d_0, t_0, p_0, d_1, t_1, p_1)$ . Again, the number of turns of a history is defined as the length of this path, each arc counting for  $1/3$ . So the number of turns



of  $h$  is 1 and  $2/3$ . Choosing strategies  $\delta$  and  $(\tau, \pi)$  of both players and fixing the initial state  $d_0 = \bar{d}$  determines a probability measure on the space of histories  $h$ . We denote by  $r_d^k(\delta, (\tau, \pi)) := -\mathbb{E}(S_{p_0}(\nu_0; m) + \dots + S_{p_{k-1}}(\nu_{k-1}; m))$  the expectation of the payment received by Tribune, in  $k$  turns, with respect to this measure. We also consider the *infinite horizon* or *mean payoff* game  $\Gamma^{\text{kl}}(\infty, \bar{d})$ , in which the payment of Tribune is now

$$r_d^\infty(\delta, (\tau, \pi)) = \limsup_{k \rightarrow \infty} k^{-1} r_d^k(\delta, (\tau, \pi)) .$$

We define the *value* of the game in horizon  $k$ ,  $v_d^k$ , and the value of the infinite horizon game,  $v_d^\infty$ , as well as optimal strategies, by saddle point conditions, as in Section 2.1. We have the following dynamic programming principle.

► **Proposition 7.** *The value vector  $v^k = (v_d^k)_{d \in D}$  in horizon  $k$  of the stochastic game with Kullback-Leibler payments,  $\Gamma^{\text{kl}}$ , does exist. It is determined by the relations  $v^0 = 0$ ,  $v^k = f(v^{k-1})$ ,  $k = 1, 2, \dots$ , where*

$$f_d(x) = \min_{(d,t) \in E} \max_{(t,p) \in E} \log \left( \sum_{(p,d') \in E} m_{pd'} \exp(x_{d'}) \right) , \tag{4}$$

and we have  $v_d^k = \log V_d^k$ .

The explicit form of  $f$  in (4) originates from the following expression of the Legendre-Fenchel transform of Shannon entropy, which is a classical result in convex analysis, see e.g. [25].

► **Lemma 8.** *The function  $x \mapsto \log(\sum_{1 \leq i \leq n} e^{x_i})$  is convex and it satisfies*

$$\log \left( \sum_{1 \leq i \leq n} e^{x_i} \right) = \max_{1 \leq i \leq n} \sum_{1 \leq i \leq n} \nu_i (x_i - \log \nu_i); \quad \nu_i \geq 0, \quad 1 \leq i \leq n, \quad \sum_{1 \leq i \leq n} \nu_j = 1 .$$

The following result shows that the extended entropy game  $\Gamma^{\text{ent}}$  is equivalent to the stochastic mean payoff game  $\Gamma^{\text{kl}}$ , through logarithmic glasses. Theorem 2 above is deduced from it. We define the *projection* of a pair of strategy  $(\delta, (\tau, \pi))$  in  $\Gamma^{\text{kl}}$  to be the strategy  $(\delta, \tau)$  in  $\Gamma^{\text{ent}}$ .

► **Theorem 9.** *The stochastic mean payoff game with Kullback-Leibler payments,  $\Gamma^{\text{kl}}(\infty, \cdot)$ , has a value. For all states  $d \in D$ , we have*

$$v_d^\infty = \log V_d^\infty, \quad v_d^\infty = \lim_{k \rightarrow \infty} \frac{1}{k} v_d^k .$$

Moreover, the optimal strategies of  $\Gamma^{\text{ent}}(k)$  are precisely the projections of the optimal strategies of  $\Gamma^{\text{kl}}(k)$ , for all  $k$  integer or equal to  $\infty$ .

This theorem shows that extended entropy games are particular stochastic mean payoff games (with compact action spaces). Asarin et al. [5] remarked that the special *deterministic* entropy games, in which People has only one possible action in each state, can be reencoded as deterministic mean payoff games. This can also be recovered from our approach: in this deterministic case, the simplices  $\Delta_p$  are singletons and the entropy function vanishes.

We next sketch the derivation of Theorem 9 from a result of Bolte, Gaubert and Vigeral [8] on the escape rate of nonexpansive mappings that are definable in an o-minimal structure. A map  $f$  is *nonexpansive* with respect to a norm  $\|\cdot\|$  if  $\|f(x) - f(y)\| \leq \|x - y\|$ . Recall that an o-minimal structure [12, 28] consists, for each integer  $n$ , of a family of subsets of  $\mathbb{R}^n$ . A subset of  $\mathbb{R}^n$  is said to be *definable* with respect to this structure if it belongs

to this family. It is required that definable sets are closed under the Boolean operations, under every projection map (elimination of one variable) from  $\mathbb{R}^n$  to  $\mathbb{R}^{n-1}$ , and under the lift, meaning if  $A \subset \mathbb{R}^n$  is definable, then  $A \times \mathbb{R} \subset \mathbb{R}^{n+1}$  and  $\mathbb{R} \times A \subset \mathbb{R}^{n+1}$  are also definable. It is finally required that when  $n = 1$ , definable subsets are precisely finite unions of intervals. A function  $f$  from  $\mathbb{R}^n$  to  $\mathbb{R}^k$  is said to be *definable* if its graph is definable. An important example of o-minimal structure is the *real exponential field*  $\mathbb{R}_{\text{alg,exp}}$ . The definable sets in this structure are the *subexponential sets* [28], i.e., the images under the projection maps  $\mathbb{R}^{n+k} \rightarrow \mathbb{R}^n$  of the *exponential sets* of  $\mathbb{R}^{n+k}$ , the latter being sets of the form  $\{x \mid P(x_1, \dots, x_{n+k}, e^{x_1}, \dots, e^{x_{n+k}}) = 0\}$  where  $P$  is a real polynomial. A theorem of Wilkie [29] implies that  $\mathbb{R}_{\text{alg,exp}}$  is o-minimal, see [28].

If  $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$  is nonexpansive in any norm, given any  $0 < \alpha < 1$ , we define the *discounted value vector*  $z_\alpha \in \mathbb{R}^n$  by  $f(\alpha z_\alpha) = z_\alpha$ . This vector exists and is unique (apply Banach fixed point theorem to the contraction mapping  $x \mapsto f(\alpha x)$ ).

► **Theorem 10** ([8]). *Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$  be nonexpansive in any norm, and suppose that  $f$  is definable in an o-minimal structure. Then, the limit  $\lim_{k \rightarrow \infty} f^k(0)/k$  does exist, and it coincides with the limit  $\lim_{\alpha \rightarrow 1^-} (1 - \alpha)z_\alpha$ .*

The vector  $z_\alpha$  is nothing but the value of the *discounted variant* of the stochastic game  $\Gamma^{\text{kl}}$ , where  $\alpha$  is the discount factor. The map  $f$  in (4) is nonexpansive in the sup-norm, and it is definable in the real exponential field. So Theorem 10 can be applied to it. The existence of the limit in Theorem 9 is deduced from this result.

A policy in a discounted game is said to be *Blackwell optimal* if it is optimal for all discount factors sufficiently close to one. The existence of Blackwell optimal policies is a basic feature of perfect information zero-sum stochastic games with finite action spaces (see [24, Chap. 10] for the one-player case, the two-player case builds on similar ideas, e.g. [15, Lemma 26]). It allows one to reduce the mean payoff problem to the discounted problem. We next show that this result has an analogue for entropy games. We shall say that a pair of strategies  $(\delta, \tau) \in \mathcal{P}_D \times \mathcal{P}_T$  is *Blackwell optimal* if there is a real number  $0 < \alpha_0 < 1$  such that, for all  $\alpha \in (\alpha_0, 1)$ ,  $(\delta, \tau)$  is the projection of a pair of optimal policies  $(\delta, (\tau, \pi))$  in the discounted version of the game  $\Gamma^{\text{kl}}$ . The fact that the value of the entropy games commutes with maxima and minima of policies (Theorem 3) is derived by combining Theorem 10 with the following result, whose proof relies, again, on an o-minimality argument.

► **Theorem 11.** *The stochastic perfect information game with Kullback-Leibler payments,  $\Gamma^{\text{kl}}$ , has Blackwell optimal strategies.*

#### 4 Applying the Collatz-Wielandt theorem to entropy games

The classical Collatz-Wielandt formulæ provide the following characterizations of the spectral radius  $\rho(M)$  of a nonnegative matrix  $M$ :

$$\rho(M) = \inf\{\lambda > 0 \mid \exists X \in \text{int } \mathbb{R}_+^D, MX \leq \lambda X\} = \max\{\lambda \geq 0 \mid \exists X \in \mathbb{R}_+^D \setminus \{0\}, MX = \lambda X\},$$

where  $\mathbb{R}_+^D$  denotes the nonnegative orthant of  $\mathbb{R}^D$ , and  $\text{int } \mathbb{R}_+^D$  its interior, i.e., the set of positive vectors. This has been extended to non-linear, order preserving and continuous self-maps of the standard positive cone [22, 2]. In particular, the following result can be derived from the non-linear Collatz-Wielandt formulæ in these works.

► **Theorem 12** (Corollary of [2]). *The value  $\bar{V}^\infty$  of the original entropy game  $\Gamma^{\text{ent}}$  (with a free initial state) coincides with any of the following expressions*

$$\inf\{\lambda > 0 \mid \exists X \in \text{int } \mathbb{R}_+^D, F(X) \leq \lambda X\} \quad (5)$$

$$\max\{\lambda > 0 \mid \exists X \in \mathbb{R}_+^D \setminus \{0\}, F(X) = \lambda X\} \quad (6)$$

$$\max\{\lambda > 0 \mid \exists X \in \mathbb{R}_+^D \setminus \{0\}, F(X) \geq \lambda X\}, \quad (7)$$

where  $F$  is the dynamic programming operator (2).

The value of these expressions is called the non-linear spectral radius of  $F$ . The Collatz-Wielandt formulæ are helpful to establish strong duality results, like (3), see also [1] for an application to mean payoff games and tropical geometry. Our main interest here lies in the following application of (5). We say that a state  $d$  of Despot is *significant* if the set of actions of Despot in this state,  $\{(d, t) \in E\}$ , has at least two elements (i.e., Despot has to make a choice in this state). We say that an entropy game is *Despot-free* if the Despot player does not have any significant state. A Despot-free game is essentially a one (and half) player problem, since the minimum term in the corresponding dynamic programming operator (2) vanishes. Indeed, for each  $d \in D$ , there is a unique node  $t$  such that  $(d, t) \in E$ , and we define the map  $\sigma : D \rightarrow T$  by  $\sigma(d) = t$ . The following corollary, which follows from Theorem 12 by making the change of variables  $\mu = \log \lambda$  and  $x = \log X$ , is also a special case of a result of Anantharam and Borkar [3].

► **Corollary 13.** *The logarithm of the value of a Despot-free entropy game is given by*

$$\inf \mu, \mu \in \mathbb{R}, x \in \mathbb{R}^D, \quad \mu + x_d \geq \log\left(\sum_{d' \in D} m_{p,d'} e^{x_{d'}}\right) \text{ for all } d \in D, p \in P \text{ such that } (\sigma(d), p) \in E. \quad (8)$$

## 5 Polynomial time solvability of entropy games with a few significant Despot positions

By *solving strategically* an (extended) entropy game, we mean, finding a pair of optimal policies. We assume from now that the weights  $m_{p,d}$  are integers. Since policies are combinatorial objects, solving strategically the game is a well posed problem in the Turing (bit) model of computation. Once optimal policies are known, the value of the game, which is an algebraic number, can be obtained as the Perron root of an associated integer matrix. Our main result is the following.

► **Theorem 14.** *Despot-free entropy games can be solved strategically in polynomial time.*

We indicate here the main arguments of proof.

Step 1. *Reduction to the irreducible case.* First, we associate to a Despot-free extended entropy game a projected digraph  $\bar{G}$ , with node set  $D$  and an arc  $d \rightarrow d'$  if there is a path  $(d, t, p, d')$  in the original digraph  $G$ . We say that the game is *irreducible* if  $\bar{G}$  is strongly connected. It is not difficult to see that in a Despot-free extended entropy game, the value of a state  $d$  is the maximum of the value of the irreducible games corresponding to the different strongly connected components of  $\bar{G}$  to which  $d$  has access under some policy of Tribune (this is a special case of a more general known property, see [15, Th.29]). Hence, we will assume that the game is irreducible in the rest of the proof.

The following result is a consequence of the non-linear Perron-Frobenius theorem in [16].

► **Lemma 15.** *The value of an irreducible Despot-free extended entropy game is independent of the initial state. Moreover, there is a vector  $U \in \text{int } \mathbb{R}_+^D$  and a scalar  $\lambda^* > 0$  such that  $F(U) = \lambda^*U$ , and  $\lambda^*$  coincides with the value of any initial state in this game.*

Thanks to this lemma, we will speak of “value” without making explicit the initial state. We set  $W := \max_{(p,d) \in E} m_{p,d}$  and  $n := |D|$ .

Step 2. *Reduction to a convex program with bounded feasible set.* To prove Theorem 14, we apply the ellipsoid method. To do so, we must replace the convex program (8) by another convex program whose feasible set is included in a ball  $B_2(a, R)$ , (the Euclidean ball with center  $a$  and radius  $R$ ), and contains a Euclidean ball  $B_2(a, r)$ , where  $\log(R/r)$  is polynomially bounded in the size of the input. The following key lemma allows us to do so.

► **Lemma 16.** *Suppose the game is Despot-free and irreducible. Then, the value  $\lambda^*$  of the game is such that  $1 \leq \lambda^* \leq nW$ . Moreover, there exists a vector  $U \in \text{int } \mathbb{R}_+^n$  such that  $F(U) = \lambda^*U$ , and for all  $d \in D$ ,  $1 \leq U_d \leq (nW)^{n-1}$ .*

We denote by  $\mathcal{K}$  the set of pairs  $(u, \mu) \in \mathbb{R}^D \times \mathbb{R} \simeq \mathbb{R}^{n+1}$ , such that

$$f(u) \leq \mu e + u, \quad 0 \leq u_d \leq (n-1)\lceil \log(nW) \rceil, \quad 0 \leq \mu \leq \lceil \log(nW) \rceil + 2, \quad (9)$$

where  $\lceil t \rceil$  denotes the smallest integer greater than or equal to  $t$ , and  $f$  is given by (4), recalling that  $e$  denotes the unit vector of  $\mathbb{R}^n$ . By combining Corollary 13, Lemma 15 and Lemma 16, we arrive at the following result.

► **Proposition 17.** *The value of a Despot-free irreducible entropy game coincides with the exponential of the value of the convex program:  $\min \mu$ ,  $(u, \mu) \in \mathcal{K}$ . Moreover,  $B_2(a, r) \subset \mathcal{K} \subset B_2(a, R)$  where  $a = (e, \lceil \log(nW) \rceil + 1) \in \mathbb{R}^D \times \mathbb{R}$ ,  $r := 1/3$ , and  $R := 2\sqrt{D+1}(n-1)\log(nW)$ .*

Step 3. *Show the existence of a polynomial time approximate separation oracle [17] for the program of Proposition 17.* The non-trivial separating half-spaces are obtained by computing the differential of the logarithmic expressions in (8). To do so, we use the fact that the values of the logarithm and exponential function can be approximated in polynomial time [9].

Step 4. *Show that if any two policies of Tribune yield different values  $\lambda$  and  $\lambda'$ , then,  $|\lambda - \lambda'|$  is bounded below by a rational number  $\eta_{sep} > 0$  whose number of bits is polynomially bounded in the size of the input.* This relies on separation results between algebraic numbers [27], since the value of a strategy of Tribune is an eigenvalue of a  $n \times n$  matrix with integer coefficients bounded by the number  $W$ .

Step 5. *Synthesize an optimal strategy of Tribune from an approximate solution of the program in Proposition 17.*

To any policy  $\tau$  of Tribune, we associate a dynamic programming operator  $F^\tau$ , which is the self-map of  $\mathbb{R}^D$  defined by  $F_d^\tau(X) = \sum_{(\tau(\sigma(d)), d') \in E} m_{\tau(\sigma(d))d'} X_{d'}$ . In other words,  $F^\tau(X) = M^\tau X$ , where  $M^\tau = (m_{\tau(\sigma(d))d'})_{d,d' \in D}$  is a  $|D| \times |D|$  matrix with nonnegative entries.

To explain our method, we make first the restrictive assumption that for every policy  $\tau$ , the matrix  $M^\tau$  is irreducible. In particular, we can take an optimal policy  $\tau^*$ . By a standard result of Perron-Frobenius theory [6],  $M^{\tau^*}$  has a left eigenvector  $\pi$  with positive entries, associated to the maximal eigenvalue  $\lambda^{\tau^*} := \rho(M^{\tau^*})$ , called Perron root. Hence,  $\pi M^{\tau^*} = \lambda^{\tau^*} \pi$ . Since  $\tau^*$  is optimal,  $\lambda^{\tau^*} = \lambda^*$ . Moreover, by applying Lemma 16 to the linear map  $U \mapsto (M^{\tau^*})^T U$ , where  $^T$  denotes the transposition, we deduce that  $\pi_d / \pi_{d'} \leq (nW)^{n-1}$ .

For any rational number  $\epsilon > 0$ , the ellipsoid algorithm, applied to the optimization problem of Proposition 17, yields in polynomial time a vector  $u$  and a scalar  $\mu$  such that

$f(u) \leq (\log \lambda^* + \epsilon)e + u$  and  $\lambda^* \leq \exp(\mu) \leq \lambda^* \exp(\epsilon)$ . Taking  $U := (U_d)_{d \in D}$  with  $U_d := \exp(u_d)$ , we get  $F(U) \leq \lambda^* \exp(\epsilon)U$ . We choose any policy  $\underline{\tau}$  such that  $F(U) = M^{\underline{\tau}}U$ . Therefore,  $\underline{\tau}(\sigma(d))$  is chosen to be any term attaining the maximum when evaluating  $F_d(U)$ . We claim that  $\underline{\tau}$  is optimal if  $\epsilon$  is sufficiently small.

To show the latter claim, we observe that  $M^{\tau^*}U \leq F(U)$ . Moreover, for all  $d \in D$ ,  $0 \leq \pi_d(\lambda^* \exp(\epsilon)U_d - F_d(U)) \leq \pi_d(\lambda^* \exp(\epsilon)U_d - (M^{\tau^*}(U))_d) \leq \sum_{d' \in D} \pi_{d'}(\lambda^* \exp(\epsilon)U_{d'} - (M^{\tau^*}U)_{d'}) = \pi(\lambda^* \exp(\epsilon)U - M^{\tau^*}U) = \lambda^*(\exp(\epsilon) - 1)\pi U$ . Using  $\pi_d/\pi_{d'} \leq (nW)^{n-1}$  and  $U_d/U_{d'} \leq (nW)^{n-1}$ , we deduce that  $F(U) \geq \underline{\lambda}U$ , where  $\underline{\lambda} := \lambda^*[\exp(\epsilon) - (\exp(\epsilon) - 1)n(nW)^{2(n-1)}]$ . Since,  $M^{\underline{\tau}}U \geq \underline{\lambda}U$ , we have  $\rho(M^{\underline{\tau}}) \geq \underline{\lambda}$ . It follows that we can choose  $\epsilon > 0$ , with a polynomially bounded number of bits, such that  $\underline{\lambda} > \lambda^* - \eta_{\text{sep}}$ . Moreover, since  $\lambda^*$  is the maximum of the values of all the policies,  $\underline{\lambda} \leq \lambda^*$ . By definition of the separation parameter  $\eta_{\text{sep}}$ , this implies that  $\underline{\lambda} = \lambda^*$ , and so the policy  $\underline{\tau}$  of Tribune which we just constructed is optimal, showing the claim.

When some policies  $\tau$  yield a *reducible* matrix  $M^\tau$ , the synthesis of the optimal policy  $\underline{\tau}$  still exploits the same idea with an additional technicality, since we can only guarantee that the inequality  $F_d(U) \geq \underline{\lambda}U_d$  is valid for every state  $d$  such that  $\pi_d > 0$ .

Theorem 3, showing that  $V_d^\infty = \min_{\delta \in \mathcal{P}_D} V_d^\infty(\delta, \star)$ , allows us to solve an entropy game by enumerating the policies  $\delta \in \mathcal{P}_D$  and solving the Despot-free entropy game determined by each  $\delta$ . This leads to an algorithm with execution time  $(\prod_{d \in D} |E_d|)T_{\text{D-free}}$ , where the factor  $T_{\text{D-free}}$  is polynomial in the size of the input. We have in particular:

► **Corollary 18.** *Entropy games in which Despot has a fixed number of significant states can be solved strategically in polynomial time.*

## 6 Multiplicative policy iteration algorithm and comparison with the spectral simplex method of Protasov

The equivalence between extended entropy games and some special class of stochastic mean payoff games, through logarithmic glasses (see Section 3), allows us to adapt classical algorithms for one or two player zero sum games, such as the value iteration and the policy iteration algorithm. We next present a multiplicative version of the policy iteration algorithm, which follows by adapting policy iteration ideas of Hoffman and Karp [19] and Rothblum [26].

To simplify the presentation, we consider first a Despot-free entropy game. Without loss of generality, we assume that  $D = T = \{1, \dots, |T|\}$  and  $\sigma$  is the identity. Let  $F^\tau$  and  $M^\tau$ ,  $\tau \in \mathcal{P}_T$ , be defined as in the previous section. If  $M^\tau$  is irreducible, in particular if all its entries are positive,  $M^\tau$  has an eigenvector  $X^\tau > 0$ , associated to the Perron root  $\lambda^\tau := \rho(M^\tau)$ . Moreover,  $X^\tau$  is unique up to a multiplicative constant and is called a Perron eigenvector. If all the matrices  $M^\tau$ ,  $\tau \in \mathcal{P}_T$  are irreducible, one can construct the following multiplicative version of the policy iteration algorithm.

This algorithm has a dual version, in which maximization is replaced by minimization. Then, the Hoffman-Karp's idea [19] is readily adapted to the multiplicative setting: a sequence  $\delta^k$  is constructed in a similar way as  $\tau^k$  in the dual version of Algorithm 1, except that in Step 3,  $\lambda^{\delta^k}$  and  $X^{\delta^k}$  are computed by applying Algorithm 1 to the dynamic programming operator  $F^{\delta^k}$  in which the strategy of Despot is fixed to  $\delta^k$ . We call this the *multiplicative* Hoffman-Karp algorithm. A variation of the original proof shows that this algorithm, implemented in exact arithmetics, terminates and is correct if for any pair of policies of the two players, the associated transition matrix is irreducible.

In [23], Protasov introduced the Spectral Simplex Algorithm. His algorithm is a variant of Algorithm 1 in which the policy is improved only at *one* state, which is the first state  $t$

---

**Algorithm 1** Multiplicative policy Iteration for Despot-free entropy game.

---

- 1: Initialize  $k = 1$ ,  $\tau^0, \tau^1 \neq \tau^0$  randomly.
- 2: **while**  $\tau^k \neq \tau^{k-1}$  **do**
- 3:   Compute the Perron root  $\lambda^{\tau^k}$  and a Perron eigenvector  $X^{\tau^k}$  of  $M^{\tau^k}$ .
- 4:   Compute a new policy  $\tau^{k+1}$  such that, for all  $t \in T$ ,

$$\tau^{k+1}(t) \in \operatorname{argmax}_{\tau(t) \in P, \tau \in \mathcal{P}_T} F_t^\tau(X^{\tau^k}) = \operatorname{argmax}_{p \in P, (t,p) \in E} \sum_{t' \in T, (p,t') \in E} m_{p,t'} X_{t'}^{\tau^k},$$

and set  $\tau^{k+1}(t) = \tau^k(t)$  if this choice is compatible with the former condition.

- 5:    $k \leftarrow k + 1$
  - 6: **end while**
  - 7: **return** the optimal policy  $\tau^k$ , the Perron root  $\lambda^{\tau^k}$  and Perron eigenvector  $X^{\tau^k}$  of  $M^{\tau^k}$ .
- 

■ **Table 1** Comparing multiplicative policy iteration with spectral simplex.

Number of states	10	20	30	40	50	60	70	80	90	100
Time : Policy Iteration	0.0018	0.0037	0.0057	0.0095	0.0115	0.0141	0.0171	0.0283	0.0308	0.0363
Time : Spectral Simplex-D	0.0026	0.0083	0.0158	0.0317	0.0433	0.0511	0.0797	0.1261	0.1533	0.1950
Time : Spectral Simplex	0.0034	0.0149	0.0350	0.0934	0.1419	0.1615	0.3070	0.5835	0.7418	1.0257
Iterations : Policy Iteration	3	3	3.4	3	3	3	3.2	3.2	3	3.2
Iterations : Spectral Simplex-D	5.6	7.4	10.2	10	11.8	13.4	14.8	14.2	16	17.2
Iterations : Spectral Simplex	15.4	22	40.8	53.4	57.8	83	87	102.2	106.8	122.8

such that  $F_t(X^{\tau^k}) > \lambda^{\tau^k} X_t^{\tau^k}$ . We also considered another version of Algorithm 1, in which we change the policy at only one state  $t$ , which maximizes the expression  $F_t(X^{\tau^k}) - \lambda^{\tau^k} X_t^{\tau^k}$ . We shall refer to this algorithm as ‘‘Spectral Simplex-D’’ since this is analogous to Dantzig’s pivot rule in the original simplex method [30].

We next report numerical experiments in the case of Despot-free entropy game, in order to compare Protasov’s spectral simplex algorithm (with the improvement of Dantzig’ pivot rule) with the multiplicative Policy Iteration algorithm (Algorithm 1). In Table 1, these algorithms are respectively named ‘‘Policy Iteration’’, ‘‘Spectral Simplex’’ and ‘‘Spectral Simplex-D’’.

We constructed random Despot-free instances in which  $D = T$  has cardinal  $n$ , and every coordinate of the operator is of the form  $F_t(X) = \max_{1 \leq p \leq q} \sum_{t'} A_{tt'}^p X_{t'}$ , where  $(A_{tt'}^p)$  is a 3-dimensional tensor whose entries are independent random variables drawn with the uniform law in  $[0, 1]$ . All the results below are the average made over 10 simulations, they concern the situation in which the number of actions  $q$  is kept constant, equal to 10, whereas  $n$  varies. The time is given in seconds. The number of ‘‘iterations’’ denotes the number of times that the algorithm goes through the main loop, regardless of how many operations are performed inside the loop. The computations were performed on Matlab R2016a, using an Intel(R) Core(TM) i7-6500 CPU @ 2.59GHz processor with 12,0Go of RAM.

Spectral Simplex-D appears to be more efficient than the Spectral Simplex algorithm with its original rule [23]. Both algorithms are experimentally outperformed by policy iteration, by one order of magnitude, when  $n \rightarrow \infty$ .

## 7 Concluding remarks

We developed an operator approach for entropy games, relating them with risk sensitive control via non-linear Perron-Frobenius theory. This leads to a theoretical result (polynomial time solvability of the Despot-free case), and this allows to adapt policy iteration to these games. Several issues concerning policy iteration in the spectral setting remains unsolved. A first issue is to understand what kind of approximate eigenvalue algorithms are best suited. A second issue is to identify significant classes of entropy games on which the Hoffman-Karp type policy iteration algorithm can be shown to run in polynomial time (compare with [30, 18] in the case of Markov decision processes).

**Acknowledgments.** We thank all the referees for their comments. We thank especially one referee for detailed suggestions and for pointing out reference [10].

---

### References

- 1 M. Akian, S. Gaubert, and A. Guterman. Tropical polyhedra are equivalent to mean payoff games. *International Journal of Algebra and Computation*, 22(1):125001 (43 pages), 2012. arXiv:0912.2462, doi:10.1142/S0218196711006674.
- 2 M. Akian, S. Gaubert, and R. Nussbaum. A Collatz-Wielandt characterization of the spectral radius of order-preserving homogeneous maps on cones, 2011. URL: <https://arxiv.org/abs/1112.5968>.
- 3 V. Anantharam and V.S. Borkar. A variational formula for risk-sensitive reward, 2015. arXiv:1501.00676.
- 4 D. Andersson and P.B. Miltersen. The complexity of solving stochastic games on graphs. In *Proceedings of ISAAC'09*, number 5878 in LNCS. Springer, 2009.
- 5 E. Asarin, J. Cervelle, A. Degorre, C. Dima, F. Horn, and V. Kozyakin. Entropy games and matrix multiplication games. In *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, pages 11:1–11:14, 2016. doi:10.4230/LIPIcs.STACS.2016.11.
- 6 A. Berman and R. J. Plemmons. *Nonnegative matrices in the mathematical sciences*. Academic Press, 1994.
- 7 V. D. Blondel and Y. Nesterov. Polynomial-time computation of the joint spectral radius for some sets of nonnegative matrices. *SIAM J. Matrix Anal.*, 31(3):865–876, 2009.
- 8 J. Bolte, S. Gaubert, and G. Viger. Definable zero-sum stochastic games. *Mathematics of Operations Research*, 40(1):171–191, 2014. arXiv:1301.1967, doi:10.1287/moor.2014.0666.
- 9 J.M. Borwein and P.B. Borwein. On the complexity of familiar functions and numbers. *SIAM Review*, 30(4):589–601, 1988.
- 10 T. Chen and T. Han. On the complexity of computing maximum entropy for markovian models. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, pages 571–583, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.571.
- 11 M. D. Donsker and S. R. S. Varadhan. On a variational formula for the principal eigenvalue for operators with maximum principle. *Proc. Nat. Acad. Sci. USA*, 72(3):780–783, 1975.
- 12 L. van den Dries. *Tame topology and o-minimal structures*, volume 248 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Cambridge, 1998. doi:10.1017/CB09780511525919.

- 13 W. H. Fleming and D. Hernández-Hernández. Risk-sensitive control of finite state machines on an infinite horizon. I. *SIAM J. Control Optim.*, 35(5):1790–1810, 1997. doi:10.1137/S0363012995291622.
- 14 W. H. Fleming and D. Hernández-Hernández. Risk-sensitive control of finite state machines on an infinite horizon. II. *SIAM J. Control Optim.*, 37(4):1048–1069 (electronic), 1999. doi:10.1137/S0363012997321498.
- 15 S. Gaubert and J. Gunawardena. A non-linear hierarchy for discrete event dynamical systems. In *Proc. of the Fourth Workshop on Discrete Event Systems (WODES98)*, pages 249–254, Cagliari, Italy, 1998. IEE.
- 16 S. Gaubert and J. Gunawardena. The Perron-Frobenius theorem for homogeneous, monotone functions. *Trans. of AMS*, 356(12):4931–4950, 2004. arXiv:math.FA/0105091, doi:10.1090/S0002-9947-04-03470-1.
- 17 M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.
- 18 T. D. Hansen, P. B. Miltersen, and U. Zwick. Strategy iteration is strongly polynomial for 2-player turn-based stochastic games with a constant discount factor. In *Innovations in Computer Science 2011*, pages 253–263. Tsinghua University Press, 2011.
- 19 A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Management Science. Journal of the Institute of Management Science. Application and Theory Series*, 12:359–370, 1966.
- 20 V. Kozyakin. Hourglass alternative and the finiteness conjecture for the spectral characteristics of sets of non-negative matrices. arXiv:1507.00492, 2015.
- 21 M. Lothaire. *Applied Combinatorics on Words*. Cambridge, 2005.
- 22 R. D. Nussbaum. Convexity and log convexity for the spectral radius. *Linear Algebra Appl.*, 73:59–122, 1986.
- 23 V. Yu. Protasov. Spectral simplex method. *Mathematical Programming*, 2015. doi:10.1007/s10107-015-0905-2.
- 24 M. L. Puterman. *Markov Decision Processes*. Wiley, 2005.
- 25 R. T. Rockafellar and R. J.-B. Wets. *Variational analysis*. Springer-Verlag, Berlin, 1998. doi:10.1007/978-3-642-02431-3.
- 26 U. G. Rothblum. Multiplicative markov decision chains. *Mathematics of Operations Research*, 9(1):6–24, 1984.
- 27 S. M. Rump. Polynomial minimum root separation. *Mathematics of Computation*, 145(33):327–336, 1979.
- 28 L. van den Dries. o-minimal structures and real analytic geometry. In *Current developments in mathematics, 1998 (Cambridge, MA)*, pages 105–152. Int. Press, Somerville, MA, 1999.
- 29 A. J. Wilkie. Model completeness results for expansions of the ordered field of real numbers by restricted Pfaffian functions and the exponential function. *J. Amer. Math. Soc.*, 9(4):1051–1094, 1996. doi:10.1090/S0894-0347-96-00216-0.
- 30 Y. Ye. The simplex and policy-iteration methods are strongly polynomial for the markov decision problem with a fixed discount rate, 2011. doi:10.1287/moor.1110.0516.



# Parameterized Complexity of Small Weight Automorphisms\*

Vikraman Arvind<sup>1</sup>, Johannes Köbler<sup>2</sup>, Sebastian Kuhnert<sup>3</sup>, and Jacobo Torán<sup>4</sup>

1 Institute of Mathematical Sciences (HBNI), Chennai, India  
arvind@imsc.res.in

2 Institut für Informatik, Humboldt-Universität zu Berlin, Berlin, Germany  
koebler@informatik.hu-berlin.de

3 Institut für Informatik, Humboldt-Universität zu Berlin, Berlin, Germany  
kuhnert@informatik.hu-berlin.de

4 Institut für Theoretische Informatik, Universität Ulm, Ulm, Germany  
toran@uni-ulm.de

---

## Abstract

We show that checking if a given hypergraph has an automorphism that moves exactly  $k$  vertices is fixed parameter tractable, using  $k$  and additionally either the maximum hyperedge size or the maximum color class size as parameters. In particular, it suffices to use  $k$  as parameter if the hyperedge size is at most polylogarithmic in the size of the given hypergraph.

As a building block for our algorithms, we generalize Schweitzer's FPT algorithm [ESA 2011] that, given two graphs on the same vertex set and a parameter  $k$ , decides whether there is an isomorphism between the two graphs that moves at most  $k$  vertices. We extend this result to hypergraphs, using the maximum hyperedge size as a second parameter.

Another key component of our algorithm is an orbit-shrinking technique that preserves permutations that move few points and that may be of independent interest. Applying it to a suitable subgroup of the automorphism group allows us to switch from bounded hyperedge size to bounded color classes in the exactly- $k$  case.

**1998 ACM Subject Classification** F.1.3 Complexity Measures and Classes, F.2 Analysis of Algorithms and Problem Complexity

**Keywords and phrases** Parameterized algorithms, hypergraph isomorphism.

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.7

## 1 Introduction

The Graph Automorphism problem GA asks whether a given graph has a nontrivial automorphism. We additionally require the automorphism to have small weight. The *weight* of a permutation is the number of its non-fixpoints. We are interested in the following problems:

**GA<sub>≤k</sub>**: Given a graph  $X = (V, E)$  and  $k \in \mathbb{N}$ , does  $X$  have a nontrivial automorphism that moves at most  $k$  vertices?

**GI<sub>≤k</sub>**: Given two graphs  $X_1 = (V, E_1)$  and  $X_2 = (V, E_2)$  on the same vertex set and  $k \in \mathbb{N}$ , is there an isomorphism from  $X_1$  to  $X_2$  that moves at most  $k$  vertices?

---

\* This work was supported by the Alexander von Humboldt Foundation in its research group linkage program. The second and third authors are supported by DFG grant KO 1053/7-2.



## 7:2 Parameterized Complexity of Small Weight Automorphisms

Likewise, let  $\text{GA}_{=k}$  and  $\text{GI}_{=k}$  denote the exact weight- $k$  automorphism and isomorphism problems, respectively. By  $\text{HGA}_{=k}$ ,  $\text{HGA}_{\leq k}$ ,  $\text{HGI}_{=k}$  and  $\text{HGI}_{\leq k}$  we denote the hypergraph versions of these problems.

In [12], Schweitzer showed that  $\text{GI}_{\leq k}$  parameterized by  $k$  is in FPT, giving a  $k^{\mathcal{O}(k)}$   $\text{poly}(n)$  time algorithm for it. Schweitzer's algorithm can easily be adapted to also solve  $\text{GA}_{\leq k}$ .

**Our results.** In Section 3, we generalize Schweitzer's result [12] to hypergraphs with hyperedge size bounded by  $d$ , giving  $(dk)^{\mathcal{O}(k^2)}$   $\text{poly}(N)$  time algorithms for  $\text{HGI}_{\leq k}$  and  $\text{HGA}_{\leq k}$  (throughout the paper we use  $N$  to denote the size  $|V| \cdot |E|$  of the input hypergraphs). Consequently, for hypergraphs with  $\text{poly}(\log N)$  size hyperedges the problems remain in FPT when parameterized only by  $k$ . Note that although Hypergraph Isomorphism is known to be reducible to Graph Isomorphism, there is no known reduction from  $\text{HGI}_{=k}$  to  $\text{GI}_{=k}$  (or from  $\text{HGI}_{\leq k}$  to  $\text{GI}_{\leq k}$ ).

In Section 4 we consider  $\text{HGI}_{\leq k}$  for vertex-colored hypergraphs of unbounded hyperedge size. For hypergraphs with color classes of size at most  $b$ , we obtain a  $(kb!)^{\mathcal{O}(k^2)}$   $\text{poly}(N)$  time algorithm.

In Section 6, we use color coding [1] to give FPT algorithms for  $\text{HGA}_{=k}$  parameterized by  $k$  and additionally either by the maximum hyperedge size  $d$  or by the maximum color class size  $b$ ; the runtime bounds are the same as those mentioned above. In particular, it follows that  $\text{GA}_{=k}$  parameterized only by  $k$  is in FPT. For general hypergraphs we show that  $\text{HGA}_{=k}$  is in  $\text{FPT}^{\text{GI}}$ .

In contrast to the above results, if  $X$  is a colored graph with red and blue vertices and we want to test if  $X$  has a nontrivial automorphism  $\pi$  that moves at most  $k$  blue vertices then the problem is  $\text{W}[1]$ -hard. This confirms a claim from [5, Exercise 9.02] (see also [6, Exercise 20.3.2]). As the hint given there does not work out and we require quite different ideas, we have included our proof in Section 7.

**Related work.** The parametric dual to  $\text{GA}_{\leq k}$  asks for an automorphism that has at most  $k$  fixpoints. It is NP-complete even when restricted to  $k = 0$ , where it is known as the *fixpoint free automorphism problem* [11].

A fundamental problem in algorithmic coding theory is the *minimum weight codeword problem*: Given a system of linear equations  $Ax = 0$  over  $\mathbb{F}_2$  as instance, the problem is to find the minimum weight of a nonzero solution to it. It is known to be NP-hard even to approximate to a constant factor [15, 10]. The parameterized complexity of its decision version (called **EVEN**) is also well studied [7, 2].

**EVEN.** Given a binary matrix  $A$  defining the linear code  $Ax = 0$  over  $\mathbb{F}_2$  and a parameter  $k$ , is there is a non-zero codeword of weight at most  $k$ ?

Whether **EVEN** is in FPT or  $\text{W}[1]$ -hard remains open. In contrast, **EXACTEVEN** (which asks for a codeword of weight exactly  $k$ ) is known to be  $\text{W}[1]$ -hard [7, 2]. It is interesting to compare this to our result that  $\text{GA}_{=k}$  is in FPT. A natural generalization is to consider permutation groups  $G \leq S_n$  as input, where  $G$  is given by a generating set  $S$ , and ask for a permutation in  $G$  of minimum weight. Formally, the problem of interest<sup>1</sup> is:

**PERMCODE.** Given  $S$  with  $G = \langle S \rangle \leq S_n$  and a parameter  $k \in \mathbb{N}$ , does  $G \setminus \{\text{id}\}$  contain a permutation of weight at most  $k$ ?

---

<sup>1</sup> It is called Hamming Distance Minimum Weight Problem in [4].

PERM<sub>CODE</sub> generalizes both  $\text{GA}_{\leq k}$  and EVEN. Indeed, an instance  $(X, k)$  of  $\text{GA}_{\leq k}$  can be transformed into an instance  $(S, k)$  of PERM<sub>CODE</sub> by computing a generating set  $S$  for the automorphism group of  $X$ . Computing this efficiently requires a GI oracle.

Let  $(A, k)$  be an instance of EVEN, where  $A$  has  $n$  columns. It can be reduced to an instance  $(S, 2k)$  of PERM<sub>CODE</sub> with  $G = \langle S \rangle \leq S_{2n}$  as follows: Using Gaussian elimination, compute from  $A$  an  $n \times r$  matrix  $B$  over  $\mathbb{F}_2$  whose columns generate the solution space of  $Ax = 0$ , where  $r$  is the solution-space dimension. For each column  $(b_1, \dots, b_n)^T$  of  $B$ , include in  $S$  the permutation in  $S_{2n}$  that, for  $1 \leq i \leq n$ , transposes  $2i$  and  $2i - 1$  if  $b_i = 1$  and fixes  $2i$  and  $2i - 1$  if  $b_i = 0$ .

In particular, this shows that the hardness results for EVEN also apply to PERM<sub>CODE</sub>, even when the latter is restricted to groups with orbit size 2. In Section 5, we prove that PERM<sub>CODE</sub> for a permutation group  $G$  is reducible in polynomial time to PERM<sub>CODE</sub> for a subgroup  $G'$  of  $G$  whose orbits are of size bounded by the parameter  $k$ . We use this reduction in our algorithms for  $\text{HGA}_{=k}$ .

## 2 Preliminaries

A permutation group  $G$  is a subgroup of  $\text{Sym}(V)$ , where  $\text{Sym}(V)$  is the group of all permutations on a finite set  $V$ . If  $V = [n]$  we denote  $\text{Sym}(V)$  by  $S_n$ . We denote the image of  $v \in V$  under a permutation  $\pi$  by  $v^\pi$  and sometimes also by  $\pi(v)$ . For a subset  $U \subseteq V$ , we write  $U^\pi = \pi(U) = \{u^\pi \mid u \in U\}$ . We apply permutations from left to right so that  $v^{\varphi\pi} = (v^\varphi)^\pi$ .

We write  $H \leq G$  when  $H$  is a subgroup of  $G$ . If  $\varphi$  is an element of  $G$  then  $H\varphi$  denotes the coset  $\{\pi\varphi \mid \pi \in H\}$ . For  $S \subseteq \text{Sym}(V)$ , the group  $\langle S \rangle$  generated by  $S$  is the smallest subgroup of  $\text{Sym}(V)$  containing  $S$ . For an element  $v \in V$ , the set  $\{v^\pi \mid \pi \in G\}$  is the  $G$ -orbit of  $v$ . In case  $G = \langle \{\pi\} \rangle$  we call the resulting set  $\{\pi^i(v) \mid i \in \mathbb{N}\}$  also the  $\pi$ -orbit of  $v$ .

The analysis of our algorithms relies on some measures on permutations. The support of  $\pi \in \text{Sym}(V)$  is  $\text{supp}(\pi) = \{u \in V \mid u^\pi \neq u\}$ , i.e., the set of non-fixpoints of  $\pi$ . Its complexity  $\text{compl}(\pi)$  is the size of  $\text{supp}(\pi)$  minus the number of  $\pi$ -orbits having size at least 2. Equivalently,  $\text{compl}(\pi)$  is the minimum number of transpositions whose product is  $\pi$ .

► **Definition 2.1.** Let  $G \leq \text{Sym}(V)$  and  $\pi \in \text{Sym}(V)$ ; this includes the case  $\pi = \text{id}$ .

1. A permutation  $\sigma \in G\pi \setminus \{\text{id}\}$  has *minimal support* in  $G\pi$  if there is no  $\varphi \in G\pi \setminus \{\text{id}\}$  with  $\text{supp}(\varphi) \subsetneq \text{supp}(\sigma)$ .
2. A permutation  $\sigma \in G\pi \setminus \{\text{id}\}$  has *minimal complexity* in  $G\pi$  if there are no  $\varphi \in G \setminus \{\text{id}\}$  and  $\psi \in G\pi \setminus \{\text{id}\}$  with  $\sigma = \varphi\psi$  and  $\text{compl}(\sigma) = \text{compl}(\varphi) + \text{compl}(\psi)$ , i.e., if for every way to express  $\sigma$  as the product of a minimum number of transpositions  $\sigma = \tau_1 \cdots \tau_{\text{compl}(\sigma)}$  and every  $i \in \{2, \dots, \text{compl}(\sigma)\}$  it holds that  $\tau_i \cdots \tau_{\text{compl}(\sigma)} \notin G\pi$ .

In particular, these notions apply to elements of the automorphism group  $\text{Aut}(X)$  of a graph  $X$  and to elements of the coset  $\text{Iso}(X, Y)$  of isomorphisms between two graphs  $X$  and  $Y$ .

► **Lemma 2.2.** Let  $G\pi$  be a coset of a permutation group  $G$  and let  $\sigma \in G\pi \setminus \{\text{id}\}$ . Then for some  $\ell \geq 1$  there are  $\sigma_1, \dots, \sigma_{\ell-1} \in G$  with minimal complexity in  $G$  and  $\sigma_\ell \in G\pi$  with minimal complexity in  $G\pi$  such that  $\sigma = \sigma_1 \cdots \sigma_\ell$  and  $\text{supp}(\sigma_i) \subseteq \text{supp}(\sigma)$  for each  $i \in \{1, \dots, \ell\}$ . Moreover, all these inclusions become equalities if  $\pi = \text{id}$  and  $\sigma$  has minimal support in  $G$ .

**Proof.** If  $\sigma$  has minimal complexity in  $G\pi$ , we have  $\ell = 1$  and  $\sigma_1 = \sigma$ . Otherwise, there are  $\varphi \in G \setminus \{\text{id}\}$  and  $\psi \in G\pi \setminus \{\text{id}\}$  such that  $\sigma = \varphi\psi$  and  $\text{compl}(\sigma) = \text{compl}(\varphi) + \text{compl}(\psi)$ . This implies  $\text{supp}(\varphi) \subseteq \text{supp}(\sigma)$  and  $\text{supp}(\psi) \subseteq \text{supp}(\sigma)$ , and these inclusions become equalities

if  $\pi = \text{id}$  and  $\sigma$  has minimal support in  $G$ . If  $\varphi$  and  $\psi$  do not have minimal complexity in  $G$  and  $G\pi$ , respectively, they can be decomposed further. This process terminates, as  $\varphi$  and  $\psi$  both have lower complexity than  $\sigma$ . ◀

### 3 Bounded hyperedge size

Let  $X = (V, E)$  and  $Y = (V, E')$  be hypergraphs such that for each  $e \in E$  we have  $|e| \leq d$ . We show that a nontrivial element  $\pi \in \text{Iso}(X, Y)$  of weight at most  $k$ , if it exists, can be found in  $(dk)^{\mathcal{O}(k)} \text{poly}(N)$  time.

This generalizes Schweitzer's result [12] shown for usual graphs, to hypergraphs of bounded hyperedge size. In order to find an isomorphism  $\pi$  between two given graphs such that  $\pi$  has support size at most  $k$ , Schweitzer's algorithm constructs  $\pi$  by iteratively adding transpositions that bring the input graphs closer to each other. To find suitable transpositions, it explores a search tree of depth  $k$  and degree  $\binom{2k}{2}$ . In each step, it computes a *candidate set* of at most  $2k$  vertices and tries all transpositions among them. For each isomorphism  $\pi$  between the input graphs that has support size at most  $k$ , this candidate set contains two vertices that are in the same orbit of  $\pi$ . Vertex covers play a crucial role in computing the candidate set. To extend this algorithm to hypergraphs of bounded hyperedge size, we need a generalization of vertex covers.

► **Definition 3.1.** Let  $X$  be a hypergraph on a vertex set  $V$ , and let  $C \subseteq V$ . A hyperedge  $e$  of  $X$  is *q-covered* by  $C$  if  $|e \cap C| \geq \min(q, |e|)$ . The set  $C$  is a *q-strong vertex cover* of  $X$  if it *q-covers* every hyperedge of  $X$ .

► **Definition 3.2.** Let  $X$  and  $Y$  be hypergraphs on a vertex set  $V$ .  $X \triangle Y$  is the hypergraph with edge set  $E(X) \triangle E(Y)$  and having as vertex set the union of all edges in  $E(X) \triangle E(Y)$ .

For the following results, let  $X$  and  $Y$  be two non-identical hypergraphs on the same vertex set  $V$  and let  $\pi$  be an isomorphism from  $X$  to  $Y$  with  $|\text{supp}(\pi)| \leq k$ .

► **Lemma 3.3.** *Let  $C$  be a q-strong vertex cover of  $X \triangle Y$  such that no two distinct points in  $C$  belong to the same  $\pi$ -orbit. Then for no hyperedge  $e$  of  $X \triangle Y$  with  $|e \cap C| \leq q$  it holds that  $e \cap \text{supp}(\pi) \subseteq C$ .*

**Proof.** Let  $e = \{u_1, \dots, u_t\} \in E(X) \triangle E(Y)$  be a hyperedge such that  $u_1, \dots, u_s \in C$  and  $u_{s+1}, \dots, u_t \notin C$  for some  $s \leq q$ . Suppose, to the contrary, that  $e \cap \text{supp}(\pi) \subseteq C$ , i.e.,  $u_i^\pi = u_i$  for  $i \in \{s+1, \dots, t\}$ . W.l.o.g. let  $e \in E(X) \setminus E(Y)$ . Let  $\ell$  be the length of the  $\pi$ -orbit of  $e$ . It is the least positive integer such that  $\pi^\ell(e) = e$ . We claim that  $\pi^{i-1}(e) \in E(X)$  implies  $\pi^i(e) \in E(X)$ , for  $1 \leq i < \ell$ . This contradicts  $\pi^{\ell-1}(e) \notin E(X)$ , which follows from  $\pi^\ell(e) = e \notin E(Y)$  because  $\pi$  is an isomorphism from  $X$  to  $Y$ .

To prove the claim, fix any  $j \in \{1, \dots, s\}$  with  $\pi^i(u_j) \neq u_j$ . Such a  $j$  must exist because otherwise  $\pi^i(e) = e$  for  $i < \ell$ , contradicting the definition of  $\ell$ . As  $u_j \in C$  and no two distinct points in  $C$  belong to the same  $\pi$ -orbit, it follows that  $\pi^i(u_j) \notin C$ , implying that  $\pi^i(e)$  is not *q-covered* by  $C$  and thus cannot be contained in  $E(X) \triangle E(Y)$ . Finally, as  $\pi$  is an isomorphism from  $X$  to  $Y$ ,  $\pi^{i-1}(e) \in E(X)$  implies that  $\pi^i(e) \in E(Y)$ , but since  $\pi^i(e) \notin E(X) \triangle E(Y)$ , it follows that  $\pi^i(e) \in E(X)$ . ◀

► **Lemma 3.4.** *If  $C$  is a q-strong vertex cover of  $X \triangle Y$  such that no two distinct points in  $C$  belong to the same  $\pi$ -orbit, then  $C \cup \text{supp}(\pi)$  is a  $(q+1)$ -strong vertex cover of  $X \triangle Y$ .*

**Proof.** For all hyperedges  $e$  of  $X \triangle Y$  with  $|e \cap C| \leq q$  we have  $e \cap \text{supp}(\pi) \not\subseteq C$  by Lemma 3.3. Hence,  $\text{supp}(\pi)$  covers at least one additional vertex in each such hyperedge. ◀

■ **Algorithm 1**  $\text{CandidateSet}_{k,d}(X \triangle Y)$

```

1 Input: The symmetric difference  $X \triangle Y$  of two hypergraphs  $X \neq Y$  on vertex
2   set  $V = [n]$  containing some hyperedge of size at most  $d$ 
3 Output: A candidate set  $C$  of size at most  $dk$  that contains for any isomorphism
4    $\pi \in \text{Iso}(X, Y)$  with  $|\text{supp}(\pi)| \leq k$  two elements belonging to the same  $\pi$ -orbit
5  $C_0 \leftarrow \emptyset; q \leftarrow 0$ 
6 while  $X \triangle Y$  has a  $(q + 1)$ -strong vertex cover  $C_{q+1} \supseteq C_q$  with  $|C_{q+1}| \leq |C_q| + k$ 
7   and  $q < d$  do  $q \leftarrow q + 1$ 
8 return  $C_q$ 

```

► **Lemma 3.5.** *If  $C$  is a  $q$ -strong vertex cover for  $X \triangle Y$  and some hyperedge  $e$  of  $X \triangle Y$  has size at most  $q$ , then  $C$  contains two distinct points belonging to the same  $\pi$ -orbit.*

**Proof.** As  $C$  is a  $q$ -strong vertex cover and  $e$  has size at most  $q$  it follows that  $C$  contains  $e$ . If no two distinct points in  $C$  belong to the same  $\pi$ -orbit, then Lemma 3.3 implies that  $C$  does not even contain  $e \cap \text{supp}(\pi)$ , a contradiction. ◀

Lemmas 3.4 and 3.5 suggest the following algorithm for computing a candidate set; it can be plugged into Schweitzer’s isomorphism test [12, Algorithm 1], cf. Algorithm 2 below.

Observe that, by construction, Algorithm 1 returns a candidate set  $C_q$  of size at most  $dk$ . The following lemma shows that  $C_q$  also has the other desired property and gives an FPT bound on the running time of the procedure.

► **Lemma 3.6.** *If  $X, Y$  are hypergraphs such that  $X \triangle Y$  has an hyperedge of size at most  $d$ , and  $\pi$  is an isomorphism from  $X$  to  $Y$  with  $|\text{supp}(\pi)| \leq k$ , then the set  $C_q$  returned by  $\text{CandidateSet}_{k,d}(X \triangle Y)$  contains two vertices in the same  $\pi$ -orbit. Moreover, the running time of the procedure is  $\mathcal{O}(d^{k+1} \text{poly}(N))$ , where  $N$  is the length of the encoding of  $X \triangle Y$ .*

**Proof.** Observe that  $C_0 = \emptyset$  is a 0-strong vertex cover. Lemma 3.4 guarantees that for  $q < d$ , the condition of the while-loop can only be violated if  $C_q$  contains two vertices in the same  $\pi$ -orbit. Furthermore, Lemma 3.5 guarantees that this also holds in the case that  $q$  reaches the value  $d$ .

It remains to show the bound on the running time. The only critical step is to extend a  $q$ -strong vertex cover  $C_q$  of  $X \triangle Y$  to a  $(q + 1)$ -strong one  $C_{q+1}$  in Line 6. This can be reduced to finding a hitting set  $S$  of size at most  $k$  for the hypergraph  $\{e \in \text{E}(X) \triangle \text{E}(Y) \mid e \setminus C_q \neq \emptyset \wedge |e \cap C_q| = q\}$  and taking  $C_{q+1} = C_q \cup S$ . The latter problem is fixed parameter tractable by the classical bounded search tree technique in time  $\mathcal{O}(d^k \text{poly}(N))$  (see, e.g., [8, Theorem 1.14]). ◀

The following is a search version of Schweitzer’s algorithm adapted to hypergraphs.

Finding *all* isomorphisms of support size at most  $k$  is not possible in FPT time; e.g. between two complete graphs there are  $\Omega(n^k)$  of them. However, the following lemma shows that Algorithm 2 finds a meaningful subset of them.

► **Lemma 3.7.** *Let  $X \neq Y$  be two hypergraphs on the vertex set  $V$  with hyperedge size bounded by  $d$ , and let  $c, k \in \mathbb{N}$ . Then the set returned by  $\text{ISO}_{k,d}(X, Y, c)$  is a subset of  $\text{Iso}(X, Y)$  containing every complexity-minimal isomorphism  $\varphi$  from  $X$  to  $Y$  with  $|\text{supp}(\varphi)| \leq k$  and  $\text{compl}(\varphi) \leq c$ . Further,  $\text{ISO}_{k,d}(X, Y, c)$  runs in time  $\mathcal{O}((dk)^{\mathcal{O}(ck)} \text{poly}(N))$ , where  $N$  is the length of the encodings of  $X$  and  $Y$ .*

■ **Algorithm 2**  $\text{ISO}_{k,d}(X, Y, c)$

```

1 Input: Two hypergraphs  $X$  and  $Y$  on vertex set  $V = [n]$  with hyperedge size bounded
2   by  $d$  and a natural number  $c \leq k$  that bounds the recursion depth
3 Output: A set  $P$  of isomorphisms from  $X$  to  $Y$ 
4 if  $X \triangle Y$  is empty then return  $\{\text{id}\}$ 
5  $P \leftarrow \emptyset$ 
6 if  $c > 0$  then
7    $C \leftarrow \text{CandidateSet}_{k,d}(X \triangle Y)$ 
8   foreach  $v_1, v_2 \in C$  with  $v_1 \neq v_2$  do
9      $P' \leftarrow \text{ISO}_{k,d}(X, Y^{(v_1 v_2)}, c - 1)$ 
10     $P \leftarrow P \cup \{\varphi' \cdot (v_1 v_2) \mid \varphi' \in P'\}$  // compose with  $(v_1 v_2) \in \text{Iso}(Y^{(v_1 v_2)}, Y)$ 
11 return  $\{\varphi \in P \mid |\text{supp}(\varphi)| \leq k\}$ 

```

**Proof.** Clearly, the set returned by  $\text{ISO}_{k,d}(X, Y, c)$  only contains isomorphisms from  $X$  to  $Y$ .

It remains to show that every complexity-minimal isomorphism  $\varphi$  from  $X$  to  $Y$  with  $|\text{supp}(\varphi)| \leq k$  and  $\text{compl}(\varphi) \leq c$  is in this set. By Lemma 3.6, the candidate set  $C$  contains two vertices  $v_1$  and  $v_2$  that belong to the same orbit of  $\varphi$ . Thus we get  $\varphi = \varphi' \cdot (v_1 v_2)$  for some  $\varphi'$  with  $\text{compl}(\varphi') = \text{compl}(\varphi) - 1$ . Note that if  $\varphi' \neq \text{id}$  then  $\varphi'$  has minimal complexity in  $\text{Iso}(X, Y^{(v_1 v_2)})$ . Indeed,  $\varphi' = \varphi_1 \varphi'_2$  with  $\varphi_1 \in \text{Aut}(X) \setminus \{\text{id}\}$ ,  $\varphi'_2 \in \text{Iso}(X, Y^{(v_1 v_2)})$  and  $\text{compl}(\varphi') = \text{compl}(\varphi_1) + \text{compl}(\varphi'_2)$  would imply  $\varphi = \varphi_1 \varphi_2$  with  $\varphi_1 \in \text{Aut}(X) \setminus \{\text{id}\}$ ,  $\varphi_2 = \varphi'_2 \cdot (v_1 v_2) \in \text{Iso}(X, Y)$  and  $\text{compl}(\varphi) = \text{compl}(\varphi_1) + \text{compl}(\varphi_2)$ , contradicting that  $\varphi$  has minimal complexity in  $\text{Iso}(X, Y)$ .

Now it suffices to show that when  $v_1$  and  $v_2$  are considered in the loop starting in Line 8, the permutation  $\varphi'$  is contained in the set returned by  $\text{ISO}_{k,d}(X, Y^{(v_1 v_2)}, c - 1)$ . This follows by induction on the complexity of  $\varphi$ , with the base case  $\varphi = \text{id}$  taken care of by Line 4.

To show the bound on the running time, it suffices to observe that the depth of the recursion tree is  $c$  and, as  $|C| \leq dk$ , there are  $\mathcal{O}((dk)^{2c})$  recursive calls in total. In each call, it takes  $\mathcal{O}(d^{k+1} \text{poly}(N))$  time to compute  $C$  (Lemma 3.6), and the rest of the work is linear in the size of the recursion tree. ◀

We remark that an isomorphism  $\varphi$  in the set  $P$  returned by  $\text{ISO}_{k,d}$  has minimal complexity if and only if there is no  $\sigma \in P$  with  $\text{compl}(\sigma) < \text{compl}(\varphi)$  and  $\text{compl}(\varphi) = \text{compl}(\varphi\sigma^{-1}) + \text{compl}(\sigma)$ ; note that this condition can be checked in polynomial time.

► **Theorem 3.8.** *Given two hypergraphs  $X \neq Y$  with hyperedge size bounded by  $d$ , it can be decided in time  $\mathcal{O}((dk)^{\mathcal{O}(k^2)} \text{poly}(N))$  whether there is an isomorphism from  $X$  to  $Y$  of support size at most  $k$ , where  $N$  is the length of the encodings of  $X$  and  $Y$ .*

**Proof.** The algorithm runs  $\text{ISO}_{k,d}(X, Y, k)$  and accepts if the returned set is not empty. Every isomorphism  $\varphi$  from  $X$  to  $Y$  with  $|\text{supp}(\varphi)| \leq k$  trivially satisfies  $\text{compl}(\varphi) \leq k$  and can be decomposed by Lemma 2.2, obtaining a minimal-complexity isomorphism  $\varphi'$  from  $X$  to  $Y$  with  $\text{supp}(\varphi') \subseteq \text{supp}(\varphi)$ . By Lemma 3.7,  $\varphi'$  is in the set returned by  $\text{ISO}_{k,d}(X, Y, k)$ . ◀

We conclude this section by showing how to decide the problem  $\text{HGA}_{\leq k}$  for hypergraphs with hyperedge size bounded by  $d$  in time  $\mathcal{O}((dk)^{\mathcal{O}(k^2)} \text{poly}(N))$ .

► **Theorem 3.9.** *Given a hypergraph  $X$  on  $n$  vertices with hyperedge size bounded by  $d$ , the algorithm  $\text{AUT}_{k,d}(X)$  enumerates all complexity-minimal automorphisms of  $X$  with support size at most  $k$  (plus possibly some more that do not have minimal complexity) in  $\mathcal{O}((dk)^{\mathcal{O}(k^2)} \text{poly}(N))$  time.*

■ **Algorithm 3**  $\text{AUT}_{k,d}(X)$

```

1 Input: A hypergraph  $X$  on vertex set  $V = [n]$  with hyperedge size bounded by  $d$ .
2 Output: A set  $P$  of automorphisms of  $X$ .
3  $P \leftarrow \emptyset$ 
4 foreach  $v_1, v_2 \in V$  do
5    $P' \leftarrow \text{ISO}_{k,d}(X, X^{(v_1 v_2)}, k - 1)$ 
6    $P \leftarrow P \cup \{\varphi' \cdot (v_1 v_2) \mid \varphi' \in P'\}$  // compose with the isomorphism  $(v_1 v_2)$ 
7   from  $X^{(v_1 v_2)}$  to  $X$ 
8 return  $\{\varphi \in P \mid |\text{supp}(\varphi)| \leq k\}$ 

```

**Proof.** Clearly, all elements in the returned set are automorphisms of  $X$  with support size at most  $k$ . The fact that every complexity-minimal automorphism  $\varphi$  with  $|\text{supp}(\varphi)| \leq k$  is found follows from Lemma 3.7 using the same decomposition argument as in the proof of the latter. Also the time bound follows immediately from Lemma 3.7. ◀

We remark that there is no FPT algorithm that enumerates all automorphisms with support size at most  $k$  (including those that do not have minimal support), as e.g.  $K_n$  has  $\sum_{i=1}^k \binom{n}{k} k!$  such automorphisms, which is not an FPT number. However, by Lemma 2.2 each  $\sigma \in \text{Aut}(X) \setminus \{\text{id}\}$  with support size at most  $k$  can be written as a product of minimal-complexity automorphisms of  $X$  with support size at most  $k$ . Thus  $\sigma \in \langle S \rangle$ , where  $S$  is the set returned by  $\text{AUT}_{k,d}(X)$ .

#### 4 Bounded color class size

In this section we consider vertex colored hypergraphs using the maximum color class size  $b$  as an additional parameter. We call a colored hypergraph  $b$ -bounded if the size of each of its color classes is bounded by  $b$ . We give an FPT algorithm for  $\text{HGI}_{\leq k}$  for  $b$ -bounded hypergraphs. More precisely, given hypergraphs  $X = (V, E)$  and  $Y = (V, E')$  along with a partition  $\mathcal{C} = \{C_1, C_2, \dots, C_m\}$  of  $V$  into (pairwise disjoint) color classes  $C_i$  with  $|C_i| \leq b$ , our algorithm will compute in time  $\mathcal{O}((kb!)^{\mathcal{O}(k^2)} \text{poly}(N))$  a color-preserving isomorphism from  $X$  to  $Y$  of weight at most  $k$  (if it exists). For a permutation  $\pi \in \text{Sym}(V)$  let  $\mathcal{C}[\pi] = \{C_i \in \mathcal{C} \mid \exists v \in C_i : v^\pi \neq v\}$  be the subset of color classes that intersect  $\text{supp}(\pi)$ . Suppose  $\pi \in \text{Iso}(X, Y)$  is an isomorphism of weight at most  $k$  and that  $\mathcal{C}[\pi] = \{C_{i_1}, C_{i_2}, \dots, C_{i_\ell}\}$ ,  $\ell \leq k/2$ . In order to search for the color classes in  $\mathcal{C}[\pi]$  we will apply the color-coding method of Alon-Yuster-Zwick [1]. Consider the FKS [9] family  $\mathcal{H}$  of perfect hash functions  $h: [m] \rightarrow [\ell]$ . We can use each  $h \in \mathcal{H}$  to partition the color classes  $C_1, C_2, \dots, C_m$  into  $\ell$  bags  $\mathcal{B}_1, \dots, \mathcal{B}_\ell$ , where  $\mathcal{B}_j$  contains all color classes  $C_i$  labeled with  $h(i) = j$ . Since  $\mathcal{H}$  is a perfect family of hash functions, some  $h \in \mathcal{H}$  is good for  $\pi$  in the sense that the color classes  $C_{i_1}, \dots, C_{i_\ell}$  all have distinct labels in  $[\ell]$ , i.e.,  $\{h(i_1), h(i_2), \dots, h(i_\ell)\} = [\ell]$ .

For  $j \in [\ell]$ , we define the hypergraphs  $X_j = (V_j, E_j)$  and  $Y_j = (V_j, E'_j)$  as follows:  $V_j = \bigcup \mathcal{B}_j$ ,  $E_j = \{e \cap V_j \mid e \in E\}$ , and  $E'_j = \{e \cap V_j \mid e \in E'\}$ .

Notice that if  $h \in \mathcal{H}$  is good for the target isomorphism  $\pi \in \text{Iso}(X, Y)$ , then the restriction of  $\pi$  to  $V_j$  is an isomorphism from  $X_j$  to  $Y_j$  that moves only vertices of exactly one color class in  $\mathcal{B}_j$ . We say that a color class  $C_i \in \mathcal{B}_j$  *allows to move* a hyperedge  $e \in E$  if there is an isomorphism  $\pi \in \text{Iso}(X_j, Y_j)$  that moves only vertices of color class  $C_i$  and  $(e \cap V_j)^\pi \neq e \cap V_j$ . We denote the set of all color classes  $C_i \in \mathcal{B}_j$  that allow to move  $e$  by  $\text{Movers}_j(e)$  and define

$\text{Movers}(e) = \bigcup_{j=1}^{\ell} \text{Movers}_j(e)$ . The next claim shows that the size of  $\text{Movers}(e)$  is bounded by  $\sum_{j=1}^{\ell} \log|E'_j| \leq \ell \log|E'| = \ell \log|E|$ .

► **Claim.** *For each  $j \in [\ell]$  and each hyperedge  $e \in E$ , there at most  $\log|E'_j|$  many color classes  $C_i \in \mathcal{B}_j$  that allow to move  $e$ .*

**Proof of the claim.** Suppose there are  $t > \log|E'_j|$  many color classes  $C_{j_1}, \dots, C_{j_t} \in \mathcal{B}_j$  that allow to move  $e$ . Let  $\pi_{j_r} \in \text{Iso}(X_j, Y_j)$ , for  $r \in [t]$ , be corresponding isomorphisms such that  $\pi_{j_r}$  fixes all color classes in  $\mathcal{B}_j$  except  $C_{j_r}$  and  $(e \cap V_j)^{\pi_{j_r}} \neq e \cap V_j$ . Clearly, for each subset  $T \subseteq [t]$ , the product  $\prod_{r \in T} \pi_{j_r}$  is in  $\text{Iso}(X_j, Y_j)$  and all the images  $(\prod_{r \in T} \pi_{j_r})(e \cap V_j)$ , for  $T \subseteq [t]$ , are distinct. Hence the total number of distinct edges in  $E'_j$ , generated as such images of  $(e \cap V_j)$ , will be  $2^t > |E'_j|$  which is impossible. ◀

Now we show the main result of this section. Before proceeding we introduce a definition that is important for the rest of the paper.

► **Definition 4.1.** Let  $X = (V, E)$  and  $Y = (V, E)$  be two hypergraphs with color class set  $\mathcal{C} = \{C_1, \dots, C_m\}$  and let  $\pi \in \text{Sym}(V)$ . For a subset  $\mathcal{C}' \subseteq \mathcal{C}[\pi]$  define the permutation  $\pi_{\mathcal{C}'}$  as

$$\pi_{\mathcal{C}'}(v) = \begin{cases} v^\pi, & \text{if } v \in \bigcup \mathcal{C}', \\ v, & \text{if } v \notin \bigcup \mathcal{C}'. \end{cases}$$

An isomorphism  $\pi \neq \text{id}$  from  $X$  to  $Y$  is said to be *color-class-minimal*, if for any set  $\mathcal{C}'$  with  $\emptyset \subsetneq \mathcal{C}' \subsetneq \mathcal{C}[\pi]$ , the permutation  $\pi_{\mathcal{C}'}$  is not in  $\text{Iso}(X, Y)$ .

We notice that all isomorphisms having minimal support are also color-class-minimal. Another immediate consequence of Definition 4.1 is the following lemma for decomposing automorphisms of hypergraphs.

► **Lemma 4.2.** *Let  $X = (V, E)$  be a hypergraph with color class set  $\mathcal{C} = \{C_1, C_2, \dots, C_m\}$ . Each nontrivial automorphism  $\pi$  of  $X$  can be written as a product of nontrivial color-class-minimal automorphisms  $\pi_1, \pi_2, \dots, \pi_\ell$  of  $X$ , where the support color class sets  $\mathcal{C}[\pi_i]$ , for  $1 \leq i \leq \ell$ , are pairwise disjoint and form a partition of the support color class set  $\mathcal{C}[\pi]$ .*

We now describe an algorithm that computes isomorphisms between hypergraphs by building them color class by color class. Let  $\{\mathcal{B}_1, \dots, \mathcal{B}_\ell\}$  be a partition of the color class set  $\mathcal{C} = \{C_1, C_2, \dots, C_m\}$  and let  $k = k_1 + \dots + k_\ell$ . Then we call a permutation  $\pi \in \text{Sym}(V)$   $(k_1, \dots, k_\ell)$ -good for  $\{\mathcal{B}_1, \dots, \mathcal{B}_\ell\}$ , if each bag  $\mathcal{B}_j$  contains exactly one of the color classes in  $\mathcal{C}[\pi]$  (say  $C_{i_j}$ ) and  $|\text{supp}(\pi) \cap C_{i_j}| = k_j$  for  $j = 1, \dots, \ell$ .

The following lemma shows that Algorithm 4 finds a meaningful set of isomorphisms.

► **Lemma 4.3.** *Let  $X \neq Y$  be two  $b$ -bounded hypergraphs. Then the set returned by the algorithm  $\text{ColoredIso}_{k_1, \dots, k_\ell, b, \mathcal{B}_1, \dots, \mathcal{B}_\ell}(X, Y, \text{id})$  contains all color-class-minimal isomorphisms  $\varphi$  from  $X$  to  $Y$  that are  $(k_1, \dots, k_\ell)$ -good for  $\mathcal{B}_1, \dots, \mathcal{B}_\ell$ . Furthermore, Algorithm 4 runs in time  $\mathcal{O}((b!)^k \text{poly}(N))$ .*

**Proof.** Let  $\varphi$  be such an isomorphism, and let  $\pi = \varphi_{\mathcal{C}'}$  for some subset  $\mathcal{C}' \subsetneq \mathcal{C}[\varphi]$  of the color classes that intersect  $\text{supp}(\varphi)$ . We show by induction on the number of color classes in  $\mathcal{C}[\varphi] \setminus \mathcal{C}'$  touched by  $\text{supp}(\varphi)$  but not by  $\text{supp}(\pi)$  that  $\text{ColoredIso}_{k_1, \dots, k_\ell, b, \mathcal{B}_1, \dots, \mathcal{B}_\ell}(X, Y, \pi)$  finds  $\varphi$ . If this number is 0, we have  $\varphi = \pi$ , and Line 5 ensures that  $\varphi$  is found. Otherwise, the color-class-minimality of  $\varphi$  implies that  $\pi \notin \text{Iso}(X, Y)$ . Since  $\varphi$  is  $(k_1, \dots, k_\ell)$ -good for  $\mathcal{B}_1, \dots, \mathcal{B}_\ell$ , for any hyperedge  $e \in E(X)$  with  $e^\pi \notin E(Y)$ , there is a bag  $\mathcal{B}_j$  with  $\text{supp}(\pi) \cap \bigcup \mathcal{B}_j = \emptyset$



■ **Algorithm 4**  $\text{ColoredIso}_{k_1, \dots, k_\ell, b, \mathcal{B}_1, \dots, \mathcal{B}_\ell}(X, Y, \pi)$

```

1 Input: Hypergraphs  $X$  and  $Y$  on vertex set  $V = C_1 \cup \dots \cup C_m$  with color classes
2    $C_i$  of size  $|C_i| \leq b$  and a permutation  $\pi \in \text{Sym}(V)$ 
3 Output: A set  $P$  of color-preserving isomorphisms from  $X$  to  $Y$ 
4 if  $\pi$  is a  $(k_1, \dots, k_\ell)$ -good isomorphism from  $X$  to  $Y$  for  $\mathcal{B}_1, \dots, \mathcal{B}_\ell$  then
5   return  $\{\pi\}$ 
6 else
7    $P \leftarrow \emptyset$ 
8   pick a hyperedge  $e \in E(X)$  with  $e^\pi \notin E(Y)$ 
9   foreach bag  $\mathcal{B}_j$  with  $\text{supp}(\pi) \cap \bigcup \mathcal{B}_j = \emptyset$  do
10    foreach color class  $C \in \text{Movers}_j(e)$  do
11      foreach  $\sigma \in \text{Sym}(V)$  with  $\text{supp}(\sigma) \subseteq C$  and  $|\text{supp}(\sigma)| = k_j$  do
12         $P \leftarrow P \cup \text{ColoredIso}_{k_1, \dots, k_\ell, b, \mathcal{B}_1, \dots, \mathcal{B}_\ell}(X, Y, \pi\sigma)$ 
13 return  $P$ 

```

and a color class  $C \in \text{Movers}_j(e)$  such that  $|C \cap \text{supp}(\varphi)| = k_j$ . By the inductive hypothesis,  $\varphi$  is found in the iteration of the inner loop where  $\sigma = \varphi|_C$ .

To show the bound on the running time, it suffices to observe that the recursion tree has degree bounded by  $|\text{Movers}(e)| \cdot |\text{Sym}(C)| \leq k \log |E| \cdot b!$  and depth bounded by  $k/2$ , and that all steps in each recursive call can be implemented in  $\mathcal{O}(N)$  time. ◀

To compute all color-class-minimal isomorphisms between two  $b$ -bounded hypergraphs  $X \neq Y$  of support size exactly  $k$ , we add an initial branching over all  $\ell \in [k]$  and all FKS partitions  $\mathcal{B}_1, \dots, \mathcal{B}_\ell$  of  $C$  and trying all partitions  $k = k_1 + \dots + k_\ell$ . This adds only an extra  $k^{\mathcal{O}(k)}$  factor and yields the algorithm  $\text{ColoredIso}_{k,b}(X, Y)$  for computing all color-class-minimal isomorphisms from  $X$  to  $Y$  of support size exactly  $k$ . Further, we can also handle the case  $X = Y$ , i.e., computing all color-class-minimal automorphisms of support size exactly  $k$ , by adding another initial branching over all color classes to choose the first one that is permuted. This adds the number of vertices  $n$  as an additional factor to the running time and yields the algorithm  $\text{ColoredAut}_{k,b}(X)$  for computing all color-class-minimal automorphisms of  $X$  with support size exactly  $k$ .

► **Theorem 4.4.** *Given two  $b$ -bounded hypergraphs  $X$  and  $Y$  on vertex set  $V = [n]$  and  $k \in \mathbb{N}$ , the set of all color-class-minimal isomorphisms from  $X$  to  $Y$  with support size exactly  $k$  can be computed in  $\mathcal{O}((kb!)^{\mathcal{O}(k^2)} \text{poly}(N))$  time, where  $N$  is the size of the input hypergraphs.*

As each isomorphism having minimum support size  $k$  is also color-class-minimal, we can state the following corollary.

► **Corollary 4.5.** *There is an algorithm for  $\text{HGI}_{\leq k}$  that decides for two given  $b$ -bounded hypergraphs  $X$  and  $Y$  in time  $\mathcal{O}((kb!)^{\mathcal{O}(k^2)} \text{poly}(N))$  if there is an isomorphism from  $X$  to  $Y$  of weight at most  $k$  and computes such an isomorphism if it exists.*

## 5 Shrinking orbits while preserving small weight permutations

In this section, we show how to reduce instances of the PERMCODE problem to other instances on subgroups with orbit sizes bounded by the parameter  $k$ .

■ **Algorithm 5** PERM<sub>CODE</sub> reduction

```

1 Input: A generating set  $A$  of a group  $G = \langle A \rangle \leq S_n$  and a parameter  $k$ 
2 Output: A generating set  $B$  for a subgroup  $\langle B \rangle \leq G$  with orbits of size bounded by  $k$ 
3 while  $G$  has an orbit  $O$  of size more than  $k$  do
4     pick  $u \in O$ 
5     compute a generating set  $B$  for  $G_u$  using the Schreier-Sims algorithm [13]
6      $G \leftarrow \langle B \rangle$ 
7 return  $B$ 

```

► **Lemma 5.1.** *There is a polynomial-time algorithm that on input an instance  $(A, k)$  of PERM<sub>CODE</sub>, where  $G = \langle A \rangle \leq S_n$  and  $k \in \mathbb{N}$ , computes an equivalent instance  $(B, k)$  with  $G' = \langle B \rangle \leq G$  and the property that every orbit of  $G'$  has size bounded by  $k$ . Moreover, for every  $\pi \in G$  with  $|\text{supp}(\pi)| \leq k$  there is a  $\pi' \in G'$  with  $|\text{supp}(\pi')| = |\text{supp}(\pi)|$ .*

**Proof.** The reduction is an application of the following simple group-theoretic observation.

► **Claim.** *Let  $O$  be a  $G$ -orbit of size more than  $k$  and let  $u$  be any point in  $O$ . Then, for any element  $\pi \in G$  of support size  $|\text{supp}(\pi)| \leq k$ , there is an element  $\pi' \in G_u$  with  $|\text{supp}(\pi')| = |\text{supp}(\pi)|$  where  $G_u = \{\varphi \in G \mid u^\varphi = u\}$ .*

To prove the claim, we only have to consider the case that  $u \in \text{supp}(\pi)$ , since otherwise  $\pi \in G_u$ . Let  $v$  be a point in  $O \setminus \text{supp}(\pi)$  and let  $\sigma \in G$  such that  $v^\sigma = u$ . Then it follows immediately that the permutation  $\pi' = \sigma^{-1}\pi\sigma$  belongs to  $G_u$  and has the same support size as  $\pi$ . The claimed reduction is given by the following simple algorithm that stabilizes points in orbits of size larger than  $k$  until no such orbits exist anymore.

The correctness of the reduction is a direct consequence of the claim above. Further, the reduction is polynomial-time computable as the Schreier-Sims algorithm has polynomial running time and the while loop runs for at most  $n$  iterations. ◀

## 6 Exact support size

In this section we show that the problem  $\text{HGA}_{=k}$  of computing automorphisms of support size exactly  $k$  is also solvable in FPT for hypergraphs having hyperedges or color classes of bounded size. We will first focus on hypergraphs having hyperedges of size bounded by  $d$  and show that the  $\text{HGA}_{=k}$  problem for such graphs is FPT reducible to the  $\text{HGA}_{=k}$  problem for  $k$ -bounded hypergraphs.

We stress that Schweitzer's algorithm [12], for ordinary graphs, cannot guarantee finding isomorphisms of weight exactly  $k$ . This is because exact weight  $k$  isomorphisms (and automorphisms), unless of minimal complexity, may not get enumerated in either Schweitzer's algorithm [12] or our generalization in Section 3 to bounded hyperedge size hypergraphs.

However, each weight  $k$  automorphism is expressible as a product of automorphisms enumerated by the search. We state this as a simple corollary of Lemma 2.2 and Theorem 3.9.

► **Corollary 6.1.** *Let  $X$  be a hypergraph with hyperedges of size bounded by  $d$  and let  $S$  be the set returned by the algorithm  $\text{AUT}_{k,d}(X)$ . Then the subgroup  $G = \langle S \rangle$  of  $\text{Aut}(X)$  contains all automorphisms of  $X$  of weight at most  $k$ . In particular,  $G$  includes all automorphisms of weight exactly  $k$ .*

► **Lemma 6.2.** *Let  $X = (V, E)$  be a hypergraph with hyperedges of size bounded by  $d$ . In FPT time we can reduce the search for an exact weight  $k$  automorphism of  $X$  to finding an exact weight  $k$  automorphism of  $X$  that is vertex colored with  $k$ -bounded color classes.*

**Proof.** Let  $X = (V, E)$  be a hypergraph with hyperedges of size bounded by  $d$ . Applying Theorem 3.9, we can enumerate the set  $B$  of all complexity-minimal automorphisms of  $X$  that are of weight at most  $k$ . Clearly, any weight at most  $k$  automorphism (including the exact weight  $k$  ones) of  $X$  is in the subgroup  $G = \langle B \rangle$  of  $\text{Aut}(X)$ . Next, we can apply Lemma 5.1 to the permutation group  $G$  and replace it by the subgroup  $G'$  whose orbits are of size at most  $k$  such that if  $G$  has an exact weight  $k$  automorphism then  $G'$  also has an exact weight  $k$  automorphism.

We can designate the orbits of  $G'$  (which are all of size at most  $k$ ) as color classes of  $X$  to obtain a  $k$ -bounded hypergraph. I.e. we assign different colors to vertices that are in different orbits of  $G'$  and consider only color-preserving automorphisms. Clearly, the exact weight  $k$  automorphisms of  $X$  that survive in  $G'$  are also color-preserving automorphisms of this  $k$ -bounded colored version of hypergraph  $X$ . ◀

► **Theorem 6.3.** *There is an algorithm for  $\text{HGA}_{=k}$  for  $b$ -bounded hypergraphs  $X = (V, E)$  that decides in time  $(kb!)^{\mathcal{O}(k^2)} \text{poly}(N)$  if there is an exact weight  $k$  automorphism of  $X$  and computes such an automorphism if it exists.*

**Proof.** We apply the algorithm of Theorem 4.4 to enumerate the set  $A$  of all color-class-minimal automorphisms of  $X$  of weight at most  $k$  in time  $(kb!)^{\mathcal{O}(k^2)} \text{poly}(N)$ . By Lemma 4.2, we know that for any exact weight  $k$  color-preserving automorphism  $\pi$  of  $X$  there is an  $\ell \leq k$  such that  $\pi$  is expressible as  $\pi = \pi_1 \pi_2 \dots \pi_\ell$ , where each  $\pi_s$  is a color-class-minimal automorphism of  $X$  of weight at most  $k$  and  $\mathcal{C}[\pi_s]$ , for  $1 \leq s \leq \ell$ , forms a partition of  $\mathcal{C}[\pi]$ . I.e. each candidate  $\pi_s$  is in the enumerated set  $A$ . Let  $\mathcal{C}[\pi] = \{C_{i_1}, C_{i_2}, \dots, C_{i_r}\}$ , where  $r \leq k$  (because only  $k$  vertices are moved by  $\pi$ ).

We will again use color coding [1] to search for the  $\pi_s$ ,  $1 \leq s \leq \ell$ . Let  $\mathcal{H}$  be the FKS family of perfect hash functions  $h: [m] \rightarrow [r]$ , where  $m$  is the total number of color classes in  $X$ , and  $r \leq k$ , as above, is the number of color classes in  $\mathcal{C}[\pi]$ . For each  $j \in [r]$  define the bags  $\mathcal{B}_j = \{C_i \mid h(i) = j\}$ . By the property of the FKS family, there is some  $h$  such that for  $j = 1, \dots, r$ , each bag  $\mathcal{B}_j$  contains exactly one color class from  $\mathcal{C}[\pi]$ . Notice the following simple claim.

► **Claim.** *The total number of partitions of the set  $\{i_1, i_2, \dots, i_r\}$  into  $\ell$  sets is bounded by  $2^{r\ell}$  and we can cycle through all such partitions in time  $2^{\mathcal{O}(r\ell)}$ .*

One of the  $2^{r\ell}$  many partitions, say  $I_1 \sqcup I_2 \sqcup \dots \sqcup I_\ell$  of  $\{i_1, i_2, \dots, i_r\}$  will be the partition such that  $\mathcal{C}[\pi_s] = \{C_i \mid i \in I_s\}$ ,  $1 \leq s \leq \ell$ . Assume we are considering this partition  $I_1 \sqcup I_2 \sqcup \dots \sqcup I_\ell$ . Then in  $|A|$  time we partition  $A$  into subsets  $A_s$ , with  $1 \leq s \leq \ell$ , defined by

$$A_s = \{\psi \in A \mid C_i \in \mathcal{C}[\psi] \text{ iff } i \in I_s\}.$$

Finally, we can try all partitions  $k = k_1 + k_2 + \dots + k_\ell$  (there are at most  $2^{2k}$  many). For each partition we look for an element  $\pi_s$  of weight exactly  $k_s$  in  $A_s$  in time  $|A_s| \text{poly}(N)$ . ◀

► **Corollary 6.4.** *There is an algorithm for  $\text{HGA}_{=k}$  for hypergraphs  $X$  with hyperedges of size bounded by  $d$  that decides in time  $d^{\mathcal{O}(k)} 2^{\mathcal{O}(k^2)} \text{poly}(N)$  if there is an exact weight  $k$  automorphism of  $X$  and computes such an automorphism if it exists.*

**Proof.** By Lemma 6.2 we can transform  $X = (V, E)$  into a hypergraph with  $k$ -bounded color classes in time  $(dk)^{\mathcal{O}(k)} \text{poly}(|V|, |E|)$  such that  $X$  has an exact weight  $k$  automorphism if and only if there is an exact weight  $k$  color-preserving automorphism of  $X$ . Hence, we can apply the algorithm of Theorem 6.3 to solve the problem.  $\blacktriangleleft$

► **Remark.** For the general case of hypergraphs of unbounded edge size it is easy to see that we have an  $\text{FPT}^{\text{GI}}$  algorithm for the problem  $\text{HGA}_{=k}$  to find an exact weight  $k$  automorphism (and hence it is unlikely to be  $\text{W}[1]$ -hard). We use the GI oracle in order to first compute a generating set for  $\text{Aut}(X)$  and then using Lemma 5.1 we can reduce the instance to a  $k$ -bounded hypergraph to which Theorem 6.3 can be applied.

## 7 The Colored Graph Automorphism problem is $\text{W}[1]$ -hard

The  $\text{COLOREDGRAPHAUTOMORPHISM}$  problem [5, p. 460] is defined as follows: given a red/blue graph  $X = (\mathcal{R}, \mathcal{B}, E)$  and the parameter  $k \in \mathbb{N}$ , does  $X$  have a color-preserving automorphism whose support contains exactly  $k$  vertices in  $\mathcal{B}$  (the blue vertices)?

Exercise 9.0.2 in [5] (Exercise 20.3.2 in [6]) is to show that this problem is  $\text{W}[1]$ -hard. The book gives a reduction, as hint, from the  $\text{WEIGHTED ANTIMONOTONE 2-CNF-SAT}$  problem. The exercise is to show that the input formula has a weight  $k$  satisfying assignment if and only if the constructed graph has an automorphism that moves exactly  $2k$  blue vertices. This hint does not work: the resulting graph has no nontrivial automorphisms. However, we now present a proof for the  $\text{W}[1]$ -hardness of  $\text{COLOREDGRAPHAUTOMORPHISM}$  by giving an  $\text{FPT}$  reduction from  $\text{EXACTEVEN}$ , which is known to be  $\text{W}[1]$ -hard [7].

► **Theorem 7.1.**  $\text{EXACTEVEN}$  is  $\text{FPT}$  reducible to  $\text{COLOREDGRAPHAUTOMORPHISM}$ .

**Proof.** The reduction is based on a gadget in [3, 14] for simulating a circuit with parity gates as an instance of Graph Isomorphism. Let  $\oplus$  denote the addition in  $\mathbb{F}_2$ . We define the undirected graph  $X_2 = (V, E)$  by  $V = \{x_a, y_a, z_a \mid a \in \{0, 1\}\} \cup \{u_{a,b} \mid a, b \in \{0, 1\}\}$  and

$$E = \{(x_a, u_{a,b}), (y_b, u_{a,b}), (u_{a,b}, z_{a \oplus b}) \mid a, b \in \{0, 1\}\}.$$

This graph gadget simulates a fan-in 2 parity gate. The  $x$  and  $y$  vertices encode the inputs of the gate while the  $z$  vertices encode the output. Any automorphism in the graph mapping the input nodes corresponding to any 0-1 input values for the gate, must map the output nodes according to the value of the parity gate being simulated.

► **Lemma 7.2** ([14]). *For any  $a, b \in \{0, 1\}$ , there is a unique automorphism  $\varphi$  of  $X_2$  that maps  $x_i$  to  $x_{a \oplus i}$  and  $y_i$  to  $y_{b \oplus i}$  for  $i \in \{0, 1\}$ . Moreover, this automorphism maps  $z_i$  to  $z_{a \oplus b \oplus i}$ .*

For the simulation of a circuit with fan-in 2 parity gates, one has to construct a parity gadget for each gate, and connect by an edge the output nodes of the gadgets ( $z$  nodes) with the input nodes ( $x$  and  $y$  nodes) of the corresponding gadget as indicated in the circuit description. Any automorphism of the constructed graph mapping the input nodes as the input values of the circuit ( $x_0$  to  $x_a$  if the input value of the  $x$  gate is  $a \in \{0, 1\}$ ) must map node  $z_0$  from the output gate to  $z_b$ , where  $b$  is the output value of the circuit.

Now for the reduction from  $\text{EXACTEVEN}$ : given a system of equations  $S$ , we construct a red/blue graph  $X = (\mathcal{R}, \mathcal{B}, E)$  in the following way: For every variable  $e_i$  in the system we define two blue vertices  $e_i^0$  and  $e_i^1$  in  $\mathcal{B}$ . These are the only blue vertices in the construction. For every equation  $e_{i,1} \oplus \dots \oplus e_{i,k_i} = 0$  in  $S$  we want to translate the property that the number of variables being set to 1 in this equation has to be even. For this, we can consider

the circuit (formula) of fan-in 2 parity gates that computes the addition modulo 2 of the variables in the equation, and transform this circuit into a graph as described above. To ensure that no automorphism permutes the vertices  $z_0$  and  $z_1$  of the output gate, we add an additional neighbor to  $z_0$ . To ensure that any automorphism stabilizes the set  $\{e_i^0, e_i^1\}$  for each  $i$ , we add a vertex  $e_i$  adjacent to both  $e_i^0$  and  $e_i^1$ , and also add  $i$  additional neighbors to  $e_i$ . Now there is an assignment with exactly  $k$  ones satisfying all the equations in the system if and only if there is an automorphism in  $X$  moving exactly  $2k$  blue vertices. ◀

**Acknowledgments.** We thank the anonymous referees for their valuable comments.

---

### References

- 1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42:844–856, 1995. doi:10.1145/210332.210337.
- 2 V. Arvind, Johannes Köbler, Sebastian Kuhnert, and Jacobo Torán. Solving linear equations parameterized by Hamming weight. *Algorithmica*, 75(2):322–338, 2016. doi:10.1007/s00453-015-0098-3.
- 3 Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992. doi:10.1007/BF01305232.
- 4 Peter J. Cameron and Taoyang Wu. The complexity of the weight problem for permutation and matrix groups. *Discrete Mathematics*, 310(10):408–416, 2010. doi:10.1016/j.disc.2009.03.005.
- 5 Rod G. Downey and Michael R. Fellows. *Parameterized complexity*. Springer, New York, 1999.
- 6 Rod G. Downey and Michael R. Fellows. *Fundamentals of parameterized complexity*. Springer, London, 2013.
- 7 Rod G. Downey, Michael R. Fellows, Alexander Vardy, and Geoff Whittle. The parameterized complexity of some fundamental problems in coding theory. *SIAM Journal on Computing*, 29(2):545–570, 1999. doi:10.1137/S0097539797323571.
- 8 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin, 2006.
- 9 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with  $\mathcal{O}(1)$  worst case access time. *J. ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.
- 10 Daniele Micciancio Ilya Dumer and Madhu Sudan. Hardness of approximating the minimum distance of a linear code. *IEEE Transactions on Information Theory*, 49(1):22–37, 2003. doi:10.1109/TIT.2002.806118.
- 11 Anna Lubiw. Some NP-complete problems similar to Graph Isomorphism. *SIAM Journal on Computing*, 10(1):11–21, 1981. doi:10.1137/0210002.
- 12 Pascal Schweitzer. Isomorphism of (mis)labeled graphs. In Camil Demetrescu and Magnús M. Halldórsson, editors, *Proc. 19th ESA*, pages 370–381, Berlin, 2011. Springer. doi:10.1007/978-3-642-23719-5\_32.
- 13 Charles C. Sims. Computational methods in the study of permutation groups. In John Leech, editor, *Computational problems in abstract algebra*, pages 169–183. Pergamon Press, 1970.
- 14 Jacobo Torán. On the hardness of graph isomorphism. *SIAM Journal on Computing*, 33(5):1093–1108, 2004. doi:10.1137/S009753970241096X.
- 15 Alexander Vardy. The intractability of computing the minimum distance of a code. *IEEE Transactions on Information Theory*, 43(6):1757–1766, 1997. doi:10.1109/18.641542.



# What Can Be Verified Locally?\*

Alkida Balliu<sup>1</sup>, Gianlorenzo D'Angelo<sup>2</sup>, Pierre Fraigniaud<sup>†3</sup>, and Dennis Olivetti<sup>4</sup>

- 1 Institut de Recherche en Informatique Fondamentale (IRIF), CNRS and University Paris Diderot, Paris, France; and Gran Sasso Science Institute (GSSI), L'Aquila, Italy
- 2 Gran Sasso Science Institute (GSSI), L'Aquila, Italy
- 3 Institut de Recherche en Informatique Fondamentale (IRIF), CNRS and University Paris Diderot, Paris, France
- 4 Institut de Recherche en Informatique Fondamentale (IRIF), CNRS and University Paris Diderot, Paris, France; and Gran Sasso Science Institute (GSSI), L'Aquila, Italy

---

## Abstract

We are considering *distributed network computing*, in which computing entities are connected by a network modeled as a connected graph. These entities are located at the nodes of the graph, and they exchange information by message-passing along its edges. In this context, we are adopting the classical framework for *local distributed decision*, in which nodes must collectively decide whether their network configuration satisfies some given boolean predicate, by having each node interacting with the nodes in its vicinity only. A network configuration is accepted if and only if every node individually accepts. It is folklore that not every Turing-decidable network property (e.g., whether the network is planar) can be decided locally whenever the computing entities are Turing machines (TM). On the other hand, it is known that every Turing-decidable network property can be decided locally if nodes are running *non-deterministic* Turing machines (NTM). However, this holds only if the nodes have the ability to guess the identities of the nodes currently in the network. That is, for different sets of identities assigned to the nodes, the correct guesses of the nodes might be different. If one asks the nodes to use the same guess in the same network configuration even with different identity assignments, i.e., to perform *identity-oblivious* guesses, then it is known that not every Turing-decidable network property can be decided locally.

In this paper, we show that every Turing-decidable network property can be decided locally if nodes are running *alternating* Turing machines (ATM), and this holds even if nodes are bounded to perform identity-oblivious guesses. More specifically, we show that, for every network property, there is a local algorithm for ATMs, with at most 2 alternations, that decides that property. To this aim, we define a hierarchy of classes of decision tasks where the lowest level contains tasks solvable with TMs, the first level those solvable with NTMs, and level  $k$  contains those tasks solvable with ATMs with  $k$  alternations. We characterize the entire hierarchy, and show that it collapses in the second level. In addition, we show separation results between the classes of network properties that are locally decidable with TMs, NTMs, and ATMs. Finally, we establish the existence of completeness results for each of these classes, using novel notions of *local reduction*.

**1998 ACM Subject Classification** D.1.3 Concurrent Programming (Distributed programming), F.2.2 Nonnumerical Algorithms and Problems


**Keywords and phrases** Distributed Network Computing, Distributed Algorithm, Distributed Decision, Locality

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.8

---

\* The first, third and fourth authors received additional support from the ANR project DISPLEXITY.


† Additional support from the INRIA project GANG.

 © Alkida Balliu, Gianlorenzo D'Angelo, Pierre Fraigniaud, and Dennis Olivetti; licensed under Creative Commons License CC-BY

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 8; pp. 8:1–8:13

Leibniz International Proceedings in Informatics

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

## 1 Introduction

### 1.1 Context and objective

In the framework of network computing, *distributed decision* is the ability to check the legality of network configurations using a distributed algorithm. In this paper, we are interested in *local* distributed decision. We insist on locality, as we want the checking protocols to avoid involving long-distance communications across the network, for they are generally costly and potentially unreliable. More specifically, we consider the standard LOCAL model of computation in networks [14]. Nodes are assumed to be given distinct identities, and each node executes the same algorithm, which proceeds in synchronous rounds where all nodes start at the same time. In each round, every node sends messages to its neighbors, receives messages from its neighbors, and performs some individual computation. The model does not limit the amount of data sent in the messages, neither it limits the amount of computation that is performed by a node during a round. Indeed, the model places emphasis on the number of rounds before every node can output, as a measure of locality. (Note however that, up to some exceptions, our positive results involve messages of logarithmic size, and polynomial-time computation). A *local algorithm* is a distributed algorithm  $\mathcal{A}$  satisfying that there exists a constant  $t \geq 0$  such that  $\mathcal{A}$  terminates in at most  $t$  rounds in all networks, for all inputs. The parameter  $t$  is called the *radius* of  $\mathcal{A}$ . In other words, in every network  $G$ , and for all inputs to the nodes of  $G$ , every node executing  $\mathcal{A}$  just needs to collect all information present in the  $t$ -ball around it in order to output, where the  $t$ -ball of  $u$  is the ball  $B_G(u, t) = \{v \in V(G) : \text{dist}(u, v) \leq t\}$ .

The objective of the paper is to determine what network properties can be decided locally, as a function of the individual computing power of the nodes.

Following the guidelines of [6], we define a *configuration* as a pair  $(G, x)$  where  $G = (V, E)$  is a connected simple undirected graph, and  $x : V(G) \rightarrow \{0, 1\}^*$  is a function assigning an input  $x(u)$  to every node  $u \in V$ . A *distributed language*  $\mathcal{L}$  is a set of configurations (we consider only Turing-decidable sets). A configuration  $(G, x) \in \mathcal{L}$  is said to be *legal* w.r.t.  $\mathcal{L}$ . Note that the membership of a configuration in a distributed language is independent of the identity that may be assigned to the nodes in the LOCAL model (this is because one may want to study the same language under different computational models, including ones that assume anonymous nodes). The class LD is the set of all distributed languages that are locally decidable. That is, LD is the class of all distributed languages  $\mathcal{L}$  for which there exists a local algorithm  $\mathcal{A}$  satisfying that, for every configuration  $(G, x)$ ,

$$(G, x) \in \mathcal{L} \iff \mathcal{A} \text{ accepts } (G, x)$$

where one says that  $\mathcal{A}$  accepts if it accepts at *all* nodes. More formally, given a graph  $G$ , let  $\text{ID}(G)$  be the set of all injective functions from  $V(G)$  to positive integers, i.e.,  $\text{ID}(G)$  denote the set of all possible identity assignments to the nodes of  $G$ . Then LD is the class of all distributed languages  $\mathcal{L}$  for which there exists a local algorithm  $\mathcal{A}$  satisfying the following: for every configuration  $(G, x)$ ,

$$\begin{aligned} (G, x) \in \mathcal{L} &\Rightarrow \forall \text{id} \in \text{ID}(G), \forall u \in V(G), \mathcal{A}_{G, x, \text{id}}(u) = \text{accept} \\ (G, x) \notin \mathcal{L} &\Rightarrow \forall \text{id} \in \text{ID}(G), \exists u \in V(G), \mathcal{A}_{G, x, \text{id}}(u) = \text{reject} \end{aligned}$$

where  $\mathcal{A}_{G, x, \text{id}}(u)$  is the output of Algorithm  $\mathcal{A}$  running on the instance  $(G, x)$  with identity-assignment  $\text{id}$ , at node  $u$ . For instance, the language PROP-COL, composed of all (connected) properly colored graphs, is in LD. Similarly, the class LCL of “locally checkable labelings”,



defined in [13], satisfies  $LCL \subseteq LD$ . In fact, LCL is precisely LD restricted to configurations on graphs with constant maximum degree, and inputs of constant size.

The class NLD is the non-deterministic version of LD, i.e., the class of all distributed languages  $\mathcal{L}$  for which there exists a local algorithm  $\mathcal{A}$  *verifying*  $\mathcal{L}$ , i.e., satisfying that, for every configuration  $(G, x)$ ,

$$(G, x) \in \mathcal{L} \iff \exists c, \mathcal{A} \text{ accepts } (G, x) \text{ with certificate } c.$$

More formally, NLD is the class of all distributed languages  $\mathcal{L}$  for which there exists a local algorithm  $\mathcal{A}$  satisfying the following: for every configuration  $(G, x)$ ,

$$\begin{aligned} (G, x) \in \mathcal{L} &\Rightarrow \exists c \in \mathcal{C}(G), \forall \text{id} \in \text{ID}(G), \forall u \in V(G), \mathcal{A}_{G,x,c,\text{id}}(u) = \text{accepts} \\ (G, x) \notin \mathcal{L} &\Rightarrow \forall c \in \mathcal{C}(G), \forall \text{id} \in \text{ID}(G), \exists u \in V(G), \mathcal{A}_{G,x,c,\text{id}}(u) = \text{rejects} \end{aligned}$$

where  $\mathcal{C}(G)$  is the class of all functions  $c : V(G) \rightarrow \{0, 1\}^*$ , assigning certificate  $c(u)$  to each node  $u$ . Note that the certificates  $c$  may depend on the network and on the input to the nodes, but should be set independently of the actual identity assignment to the nodes of the network. In the following, for the sake of simplifying the notations, we shall omit specifying the domain sets  $\mathcal{C}(G)$  and  $\text{ID}(G)$  unless they are not clear from the context. It follows from the above that NLD is a class of distributed languages that can be locally *verified*, in the sense that, on legal instances, certificates can be assigned to nodes by a *prover* so that a *verifier*  $\mathcal{A}$  accepts, and, on illegal instances, the verifier  $\mathcal{A}$  rejects (i.e., at least one node rejects) systematically, and cannot be fooled by any fake certificate. For instance, the language

$$\text{TREE} = \{(G, x) : G \text{ is a tree}\}$$

is in NLD, by selecting a root  $r$  of the given tree, and assigning to each node  $u$  a counter  $c(u)$  equal to its hop-distance to  $r$ . If the given (connected) graph contains a cycle, then no counters could be assigned to fool an algorithm checking that, at each node  $u$  with  $c(u) \neq 0$ , a unique neighbor  $v$  satisfies  $c(v) < c(u)$ . In [5], NLD was proved to be exactly the class of distributed languages that are closed under lift.

Finally, [6] defined the randomized versions  $\text{BPLD}_{p,q}$  and  $\text{BPNLD}_{p,q}$ , of the aforementioned classes LD and NLD, respectively, by replacing the use of a deterministic algorithm with the use of a randomized algorithm characterized by its probability  $p$  of acceptance for legal instances, and its probability  $q$  of rejection for illegal instances. By defining  $\text{BPNLD} = \cup_{p^2+q \geq 1} \text{BPNLD}_{p,q}$ , the landscape of local decision was pictured as follows:

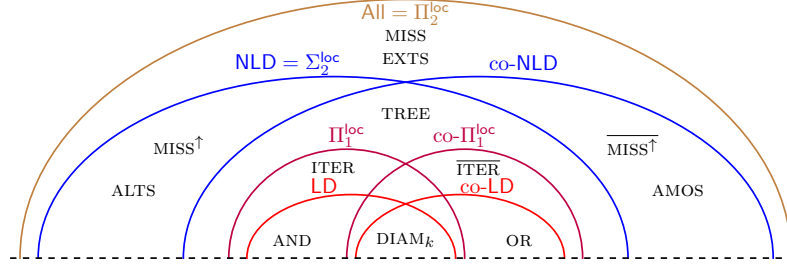
$$\text{LD} \subset \text{NLD} \subset \text{BPNLD} = \text{All}$$

where all inclusions are strict, and All is the set of all distributed languages. That is, every distributed language can be locally verified with constant success probabilities  $p$  and  $q$ , for some  $p$  and  $q$  satisfying  $p^2 + q \geq 1$ . In other words, by combining non-determinism with randomization, one can decide any given distributed language.

## 1.2 Our contributions

Following up the approach recently applied to *distributed graph automata* in [15], and to the CONGEST model in [2], we observe that the class LD and NLD are in fact the basic levels of a “local hierarchy” defined as follows. Let  $\Sigma_0^{\text{loc}} = \Pi_0^{\text{loc}} = \text{LD}$ , and, for  $k \geq 1$ , let  $\Sigma_k^{\text{loc}}$  be the class of all distributed languages  $\mathcal{L}$  for which there exists a local algorithm  $\mathcal{A}$  satisfying that, for every configuration  $(G, x)$ ,

$$(G, x) \in \mathcal{L} \iff \exists c_1, \forall c_2, \dots, \exists c_k, \mathcal{A} \text{ accepts } (G, x) \text{ with certificates } c_1, c_2, \dots, c_k$$



■ **Figure 1** Relations between the different decision classes of the local hierarchy (the definitions of the various languages can be found in the text).

where the quantifiers alternate, and  $Q$  is the universal quantifier if  $k$  is even, and the existential one if  $k$  is odd. The class  $\Pi_k^{\text{loc}}$  is defined similarly, by starting with a universal quantifier, instead of an existential one. A local algorithm  $\mathcal{A}$  insuring membership to a class  $\mathcal{C} \in \{\Sigma_k^{\text{loc}}, k \geq 0\} \cup \{\Pi_k^{\text{loc}}, k \geq 0\}$  is called a  $\mathcal{C}$ -algorithm. Hence,  $\text{NLD} = \Sigma_1^{\text{loc}}$ , and, for instance,  $\Pi_2^{\text{loc}}$  is the class of all distributed languages  $\mathcal{L}$  for which there exists a  $\Pi_2^{\text{loc}}$ -algorithm, that is, a local algorithm  $\mathcal{A}$  satisfying the following: for every configuration  $(G, x)$ ,

$$\begin{aligned} (G, x) \in \mathcal{L} &\Rightarrow \forall c_1, \exists c_2, \forall \text{id}, \forall u \in V(G), \mathcal{A}_{G,x,c_1,c_2,\text{id}}(u) = \text{accept}; \\ (G, x) \notin \mathcal{L} &\Rightarrow \exists c_1, \forall c_2, \forall \text{id}, \exists u \in V(G), \mathcal{A}_{G,x,c_1,c_2,\text{id}}(u) = \text{reject}. \end{aligned} \quad (1)$$

Our main results are the following.

► **Theorem 1.**  $\text{LD} \subset \Pi_1^{\text{loc}} \subset \text{NLD} = \Sigma_2^{\text{loc}} \subset \Pi_2^{\text{loc}} = \text{All}$ , where all inclusions are strict.

That is,  $\Pi_1^{\text{loc}} \supset \Pi_0^{\text{loc}}$ , while  $\Sigma_2^{\text{loc}} = \Sigma_1^{\text{loc}}$ , and the whole local hierarchy collapses to the second level, at  $\Pi_2^{\text{loc}}$ . In other words, while not every Turing-decidable network property can be decided locally if nodes are running *non-deterministic* Turing machines (NTM), Theorem 1 says that every Turing-decidable network property can be decided locally if nodes are running *alternating* Turing machines (ATM). More specifically, for every network property, there is a local algorithm for ATMs, with at most 2 alternations, that decides that property.

We complete our description of the local hierarchy by a collection of separation and completeness results regarding the different classes and co-classes in the hierarchy. In particular, we revisit the completeness results in [6], and show that the notion of reduction introduced in this latter paper is too strong, and may allow a language outside NLD to be reduced to a language in NLD. We introduce a more restricted form of local reduction, called *label-preserving*, which does not have this undesirable property, and we establish the following.

► **Theorem 2.** *NLD and  $\Pi_2^{\text{loc}}$  have complete distributed languages under local label-preserving reductions.*

Finally, Figure 1 summarizes all our separation results.

### 1.3 Related Work

Several form of “local hierarchies” have been investigated in the literature, with the objective of understanding the power of local computation, and/or for the purpose of designing verification mechanisms for fault-tolerant computing. In particular, [15] has investigated the case of *distributed graph automata*, where the nodes are finite automata, and the network is

anonymous (which are weaker assumptions than those in our setting), but also assuming an arbitrary global interpretation of the individual decisions of the nodes (which is a stronger assumption than those in our setting). It is shown that all levels  $\Sigma_k^{\text{aut}}$ ,  $k \geq 0$ , of the resulting hierarchy are separated, and that the whole local hierarchy is exactly composed of the MSO (monadic second order) formulas on graphs.

In the framework of distributed computing, where the computing entities are Turing machines, *proof-labeling schemes* (PLS) [8], extended to *locally checkable proofs* (LCP) [7], give the ability to certify predicates using certificates that can take benefits of the node identities. That is, for the same network predicate, and the same legal network configuration, the distributed proof that this configuration is legal may be different if the node identities are different. In this context, the whole hierarchy collapses at the first level, with  $\Sigma_1^{\text{LCP}} = \text{All}$ . However, this holds only if the certificates can be as large as  $\Omega(n^2)$  bits. In [2], the class LogLCP [7], which bounds the certificate to be of size  $O(\log n)$  bits is extended to a hierarchy that fits to the CONGEST model. In particular, it is shown that MST stands at the second level  $\Pi_2^{\text{LogLCP}}$  of that hierarchy, while there are languages outside the hierarchy.

In [6], the authors introduced the model investigated in this paper. In particular, they defined and characterized the class NLD, which is nothing else than  $\Sigma_1^{\text{loc}}$ , that is, the class of languages that have a proof-labeling scheme in which the certificates are *not* depending on the node identities. It is proved that, while  $\text{NLD} \neq \text{All}$ , randomization helps a lot, as the randomized version BPNLD of NLD satisfies  $\text{BPNLD} = \text{All}$ . It is also proved that, with the oracle  $\#\text{nodes}$  providing each node with the number of nodes in the network, we get  $\text{NLD}^{\#\text{nodes}} = \text{All}$ . Interestingly, it was proved [5] that restricting the verification algorithms for NLD to be *identity-oblivious*, that is, enforcing that each node decides the same output for every identity-assignment to the nodes in the network, does not reduce the ability to verify languages. This is summarized by the equality  $\text{NLDO} = \text{NLD}$  where the “O” in NLDO stands for identity-oblivious. In contrast, it was recently proved that restricting the algorithms to be identity-oblivious reduces the ability to decide languages locally, i.e.,  $\text{LDO} \subsetneq \text{LD}$  (see [4]).

Finally, it is worth mentioning that the ability to decide a distributed language locally has impact on the ability to design *construction* algorithms [12] for that language (i.e., computing outputs  $x$  such that the configuration  $(G, x)$  is legal w.r.t. the specification of the task). For instance, it is known that if  $\mathcal{L}$  is locally decidable, then any randomized local construction algorithm for  $\mathcal{L}$  can be derandomized [13]. This result has been recently extended [1] to the case of languages that are locally decidable by a randomized algorithm (i.e., extended from LD to BPLD according to the notations in [6]). More generally, the reader is invited to consult [3, 9, 10, 11, 14, 16] for good introductions to local computing, and/or samples of significant results related to local computing.

## 2 All languages are $\Pi_2^{\text{loc}}$ decidable

In this section, we show the last equality of Theorem 1.

► **Proposition 3.**  $\Pi_2^{\text{loc}} = \text{All}$ .

**Proof.** Let  $\mathcal{L}$  be a distributed language. We give an explicit  $\Pi_2^{\text{loc}}$ -algorithm for  $\mathcal{L}$ , i.e., a local algorithm  $\mathcal{A}$  such that, for every configuration  $(G, x)$ , Eq. (1) is satisfied. For this purpose, we describe the distributed certificates  $c_1$  and  $c_2$ . Intuitively, the certificate  $c_1$  aims at convincing each node that  $(G, x) \notin \mathcal{L}$ , while  $c_2$  aims at demonstrating the opposite. More precisely, at each node  $u$  in a configuration  $(G, x)$ , the certificate  $c_1(u)$  is interpreted as a triple  $(M(u), \text{data}(u), \text{index}(u))$  where  $M(u)$  is an  $m \times m$  boolean matrix,  $\text{data}(u)$  is a linear

array with  $m$  entries, and  $\text{index}(u) \in \{1, \dots, m\}$ . Informally,  $c_1(u)$  aims at proving to node  $u$  that it is node labeled  $\text{index}(u)$  in the  $m$ -node graph with adjacency matrix  $M(u)$ , and that the whole input data is  $\text{data}(u)$ . We denote by  $n$  the number of nodes of the actual graph  $G$ .

For a legal configuration  $(G, x) \in \mathcal{L}$ , given  $c_1$ , the certificate  $c_2$  is then defined as follows. It is based on the identification of a few specific nodes, that we call *witnesses*. Intuitively, a witness is a node enabling to demonstrate that the structure of the configuration  $(G, x)$  does not fit with the given certificate  $c_1$ . Let  $\text{dist}(u, v)$  denote the distance between any two nodes  $u$  and  $v$  in the actual network  $G$ , that is,  $\text{dist}(u, v)$  equals the number of edges of a shortest path between  $u$  and  $v$  in  $G$ . A certificate  $c_2(u)$  is of the form  $(f(u), \sigma(u))$  where  $f(u) \in \{0, \dots, 4\}$  is a flag, and  $\sigma(u) \in \{0, 1\}^*$  depends on the value of the flag.

**Case 0.** There are two adjacent nodes  $v \neq v'$  such that  $(M(v), \text{data}(v)) \neq (M(v'), \text{data}(v'))$ , or there is at least one node  $v$  in which  $c_1(v)$  cannot be read as a triple  $(M(v), \text{data}(v), \text{index}(v))$ . Then we set one of these nodes as witness  $w$ , and we set  $c_2(u) = (0, \text{dist}(u, w))$  at every node  $u$ .

Otherwise, i.e., if the pair  $(M(u), \text{data}(u))$  is identical to some pair  $(M, \text{data})$  at every node  $u$ :

**Case 1.**  $(G, x)$  is isomorphic to  $(M, \text{data})$ , preserving the inputs, denoted by  $(G, x) \sim (M, \text{data})$ , and  $\text{index}()$  represents the isomorphism. Then we set  $c_2(u) = (1)$  at every node  $u$ .

**Case 2.**  $n > m$ , i.e.,  $|V(G)|$  is larger than the dimension  $m$  of  $M$ , or  $\text{index}()$  is not injective. Then we set the certificate  $c_2(u) = (2, i, d(u, w), d(u, w'))$  where  $i \in \{1, \dots, m\}$ , and  $w \neq w'$  are two distinct nodes such that  $\text{index}(w) = \text{index}(w') = i$ . These two nodes  $w$  and  $w'$  are both witnesses.

**Case 3.**  $n < m$  and  $\text{index}()$  is injective. Then we set  $c_2(u) = (3, i)$  where  $i \in \{1, \dots, m\}$  is such that  $\text{index}(v) \neq i$  for every node  $v$ .

**Case 4.**  $n = m$  and  $\text{index}()$  is injective, but  $(G, x)$  is not isomorphic to  $(M, \text{data})$ . Then we set as witness a node  $w$  whose neighborhood in  $(G, x)$  does not fit with what it should be according to  $(M, \text{data})$ , and we set  $c_2(u) = (4, d(u, w))$  for every node  $u$ .

The local verification algorithm  $\mathcal{A}$  then proceeds as follows. First, every node  $u$  checks whether its flag  $f(u)$  in  $c_2(u)$  is identical to all the ones of its neighbors, and between 0 and 4. If not, then  $u$  rejects. Otherwise,  $u$  carries on executing the verification procedure. Its behavior depends on the value of its flag.

- If  $f(u) = 0$ , then  $u$  checks that at least one of its neighbors has a distance to the witness that is smaller than its own distance. A node with distance 0 to the witness checks that there is indeed an inconsistency with its  $c_1$  certificate (i.e., its  $c_1$  certificate cannot be read as a pair matrix-data, or its  $c_1$  certificate is distinct from the one of its neighbors). Every node accepts or rejects accordingly.
- If  $f(u) = 1$ , then  $u$  accepts or rejects according to whether  $(M(u), \text{data}(u)) \in \mathcal{L}$  (recall that, by definition, we consider only distributed languages  $\mathcal{L}$  that are Turing-decidable).
- If  $f(u) = 2$ , then  $u$  checks that it has the same index  $i$  in its certificate  $c_2$  as all its neighbors. If that is not the case, then it rejects. Otherwise, it checks each of the two distances in its certificate  $c_2$  separately, each one as in the case where  $f(u) = 0$ . A node with one of the two distances equal to 0 also checks that its  $c_1$  index is equal to the

index  $i$  in  $c_2$ . If that is not the case, or if its two distances are equal to 0, then it rejects. If all the test are passed, then  $u$  accepts.

- If  $f(u) = 3$ , then  $u$  accepts if and only if it has the same index  $i$  in its  $c_2$  certificate as all its neighbors, and  $\text{index}(u) \neq i$ .
- If  $f(u) = 4$ , then  $u$  checks the distances as in the case where  $f(u) = 0$ . A node with distance 0 also checks that its neighborhood in the actual configuration  $(G, x)$  is not what it should be according to  $(M, \text{data})$ . It accepts or rejects accordingly.

To prove the correctness of this Algorithm  $\mathcal{A}$ , let us first consider a legal configuration  $(G, x) \in \mathcal{L}$ . We show that the way  $c_2$  is defined guarantees that all nodes accept, because  $c_2$  correctly pinpoints inconsistencies in  $c_1$ , witnessing any attempt of  $c_1$  to certify that the actual configuration is illegal. Indeed, in Case 0, by the setting of  $c_2$ , all nodes but the witness accept. Also, the witness itself accepts because it does witness the inconsistency of the  $c_1$  certificate. In Case 1, all nodes accept because  $(G, x) \sim (M, \text{data})$  and  $(G, x) \in \mathcal{L}$ . In Case 2, by the setting of  $c_2$ , all nodes but the witnesses accept, and the witnesses accept too because each one checks that it is the vertex with index  $i$  in  $M$ . In Case 3, all nodes accept by construction of the certificate  $c_2$ . Finally, in Case 4, by the setting of  $c_2$ , all nodes but the witness accept. Also, the witness itself accepts because, as in Case 0, it does witness the inconsistency of the  $c_1$  certificate. So, in all cases, all nodes accept, as desired.

We are now left with the case of illegal configurations. Let  $(G, x) \notin \mathcal{L}$  be such an illegal configuration. We set  $c_1(u) = (M, \text{data}, \text{index}(u))$  where  $(M, \text{data}) \sim (G, x)$  and  $\text{index}(u)$  is the index of node  $u$  in the adjacency matrix  $M$  and the array data. We show that, for any certificate  $c_2$ , at least one node rejects. Indeed, for all nodes to accept, they need to have the same flag in  $c_2$ . This flag cannot be 1 because, if  $f(u) = 1$  then  $u$  checks the legality of  $(M, \text{data})$ . In all other cases, the distance checking should be passed at all nodes for them to accept. Thus, the flag is distinct from 0 and 4 because every radius-1 ball in  $(G, x)$  fits with its description in  $(M, \text{data})$ . Also, the flag is distinct from 2 because there are no two distinct nodes with the same index  $i$  in the  $c_1$  certificate. Finally, also the flag is distinct from 3, because, by the setting of  $c_1$ , every index in  $\{1, \dots, n\}$  appears at some node, and this node would reject. Hence, all cases lead to contradiction, that is, not all nodes can accept, as desired. ◀

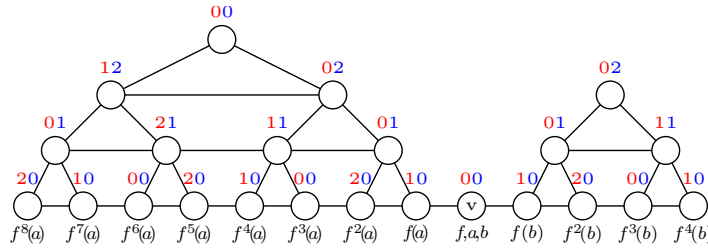
To conclude the section, let us define a simple decision task in  $\Pi_2^{\text{loc}} \setminus \text{NLD}$ . Let EXTS, which stands for “exactly two selected” be the following language. We set  $(G, x) \in \text{EXTS}$  if  $x(u) \in \{\perp, \top\}$  for every  $u \in V(G)$ , and  $|\{u \in V(G) : x(u) = \top\}| = 2$ . Proving that  $\text{EXTS} \notin \text{NLD}$  is easy using the following characterization of NLD. Let  $t \geq 1$ . A configuration  $(G', x')$  is a  $t$ -lift of a configuration  $(G, x)$  iff there exists a mapping  $\phi : V(G') \rightarrow V(G)$  that, for every  $u \in V(G')$ , induces an isomorphism between  $B_G(\phi(u), t)$  and  $B_{G'}(u, t)$ , preserving inputs (i.e.,  $x(\phi(u)) = x'(u)$  for all  $u \in V(G')$ ). A distributed language  $\mathcal{L}$  is closed under lift if there exists  $t \geq 1$  such that, for every  $(G, x)$ , we have  $(G, x) \in \mathcal{L}$  implies  $(G', x') \in \mathcal{L}$  for every  $(G', x')$  that is a  $t$ -lift of  $(G, x)$ .

► **Lemma 4** ([5]). *NLD is the class of distributed languages closed under lift.*

Since EXTS is not closed under lift, it results from Lemma 4 that  $\text{EXTS} \notin \text{NLD}$ .

### 3 On the impact of the last universal quantifier

In this section, we prove the part of Theorem 1 related to the two classes  $\Pi_1^{\text{loc}}$  and  $\Sigma_2^{\text{loc}}$ . These two classes have in common that the universal quantifier is positioned last. It results that these two classes seem to be limited, as witnessed by the following two propositions.



■ **Figure 2** An illustration of the distributed language ITER.

► **Proposition 5.**  $\Sigma_2^{\text{loc}} = \text{NLD}$ .

To show that  $\Pi_1^{\text{loc}} \neq \text{NLD}$ , we consider the language ALTS, which stands for “at least two selected”. (Note that ALTS is the complement of the language AMOS introduced in [6], where AMOS stands for “at most one selected”). We set  $(G, x) \in \text{ALTS}$  if  $x(u) \in \{\perp, \top\}$  for every node  $u \in V(G)$ , and  $|\{u \in V(G) : x(u) = \top\}| \geq 2$ . To separate NLD and  $\Pi_1^{\text{loc}}$ , we show that  $\text{ALTS} \in \text{NLD} \setminus \Pi_1^{\text{loc}}$ .

► **Proposition 6.**  $\Pi_1^{\text{loc}} \subset \text{NLD}$  (the inclusion is strict).

While  $\Pi_1^{\text{loc}}$  is in NLD, the universal quantifier adds some power compared to LD. We show that  $\text{LD} \neq \Pi_1^{\text{loc}}$  by exhibiting a language in  $\Pi_1^{\text{loc}} \setminus \text{LD}$ . Note that the existence of this language is not straightforward as it must involve Turing-computability issues. Indeed, if one does not insist on the fact that the local algorithm must be a Turing-computable function, then the two classes LD and  $\Pi_1^{\text{loc}}$  would be identical. For instance, given a  $t$ -round algorithm  $\mathcal{A}$  deciding a language  $\mathcal{L}$  in  $\Pi_1^{\text{loc}}$ , one could define the following mechanism for deciding the same language in LD. Given a  $t$ -ball  $B$  centered at  $u$ , node  $u$  accepts if and only if there are no certificate assignments to the nodes of  $B$  that could lead  $\mathcal{A}$  to reject at  $u$ . However, this mechanism is not a Turing-computable function. Interestingly, NLD would still not collapse to LD even if using non Turing-computable decision mechanisms. To see why, assume that we are given the ability to try all possible certificates of an NLD algorithm  $\mathcal{A}$ . The simple decision mechanism at every node  $u$  consisting in rejecting at  $u$  as long as  $\mathcal{A}$  rejects one of the certificates at  $u$ , which works fine for  $\Pi_1^{\text{loc}}$ , does not work for NLD. Indeed, a node that rejects a configuration for some certificate cannot safely reject because it might be a legal configuration with an incorrect certificate. We show that, in fact,  $\Pi_1^{\text{loc}} \setminus \text{LD} \neq \emptyset$ .

► **Proposition 7.**  $\text{LD} \subset \Pi_1^{\text{loc}}$  where the inclusion is strict.

**Proof.** We describe the distributed language ITER, which stands for “iteration”. Let  $M$  be a Turing machine, and let us enumerate lexicographically all the states of the system tape-machine where  $M$  starts its execution on the blank tape, with the head at the beginning of the tape. We define the function  $f_M : \mathbb{N} \rightarrow \mathbb{N}$  by  $f_M(0) = 0$ ,  $f_M(1) = 1$ , and, for  $i > 1$ ,  $f_M(i)$  equal to the index of the system state after one step of  $M$  from system state  $i$ . We define ITER as the collection of configurations  $(G, x)$  representing two sequences of iterations of a function  $f_M$  on different inputs  $a$  and  $b$  (see Figure 2).

More precisely, let  $M$  be a Turing machine, and let  $a$  and  $b$  be two non-negative integers. We define the following family of configurations (see Figure 2). A configuration in ITER mainly consists of a path  $P$  with a special node  $v$ , called the *pivot*, identified in this path. So  $P = LvR$  where  $L$  and  $R$  are subpaths, respectively called left path and right path. All nodes of the path are given the machine  $M$  as input, and the pivot  $v$  is also given  $a$  and  $b$  as inputs. The node of the left path (resp., right path) at distance  $i$  from  $v$  is given

a value  $f_{i,L}$  (resp.,  $f_{i,R}$ ) as input. To be in the language, it is required that, for every  $i$ ,  $f_{i,L} = f_M^{(i)}(a)$  and  $f_{i,R} = f_M^{(i)}(b)$ , where  $g^{(i)}$  denotes the  $i$ th iterated of a function  $g$ . Let  $u_\ell$  and  $u_r$  be the two nodes at the extremity of the left path and of the right path, respectively. The configuration is in the language if and only if the  $f$ -values at both extremities of the path  $P$  are 0 or 1, and at least one of them is equal to 0. That is, the configuration is in the language if and only if:

$$(f_{|L|,L} \in \{0, 1\} \text{ and } f_{|R|,R} \in \{0, 1\}) \text{ and } (f_{|L|,L} = 0 \text{ or } f_{|R|,R} = 0). \quad (2)$$

In fact, for technical reasons, it is also required that both  $|L|$  and  $|R|$  are powers of 2. Indeed, on top of  $L$  and  $R$  are two complete binary trees  $T_L$  and  $T_R$ , respectively, with horizontal paths connecting nodes of the same depth in each tree (see Figure 2). The nodes of  $L$  and  $R$  are the leaves of these two trees. Finally, every node  $u$  of the graph receives as input a pair of labels  $(\ell_1, \ell_2) \in \{0, 1, 2\}^2$ . The label  $\ell_1$  is the distance modulo 3 from  $u$  to the right-most node (resp., left-most node) of the path if  $u$  is an internal node of  $T_L$  (resp.,  $T_R$ ), and, for nodes in the path  $P$ ,  $\ell_1$  is simply the distance modulo 3 from the pivot  $v$ . The label  $\ell_2$  is the height of the node in its tree modulo 3. (The pivot, which belongs to none of the trees, has height 0). A configuration  $(G, x) \in \text{ITER}$  if and only if  $(G, x)$  satisfies all the above conditions with respect to the given machine  $M$ .

In other words,  $f_M$  is defined so that 1 denotes the rejecting state with any tape content, and any head position, while 0 denotes the accepting state with any tape content, and any head position. All the other configurations uniquely identify the entire tape content, the head position, and the current non halting state. In essence, when the machine switches from some configuration  $i > 1$  to another configuration  $j > 1$ , we keep track of the tape content and the head position. If the machine halts, then we discard the tape content as well as the head position, and we simply set  $f_M^{(i)}$  equal to 0 or 1 accordingly. A configuration is in the language if the machine terminates on both inputs  $a$  and  $b$ , and accepts at least one of these two inputs.

Let us consider a weaker version of  $\text{ITER}$ , denoted by  $\text{ITER}^-$  where the condition of Eq. (2) is replaced by just:  $f_{|L|,L} \in \{0, 1\}$  and  $f_{|R|,R} \in \{0, 1\}$ . Thanks to the labeling  $(\ell_1, \ell_2)$  at each node, which “rigidifies” the structure, we have  $\text{ITER}^- \in \text{LD}$  using the same arguments as the ones in [4]. Moreover,  $\text{ITER} \in \Pi_1^{\text{loc}}$ . To see why, we describe a local algorithm  $\mathcal{A}$  using certificates. The algorithm first checks whether  $(G, x) \in \text{ITER}^-$ . All nodes, but the pivot  $v$ , decide according to this checking. If the pivot rejected  $(G, x) \in \text{ITER}^-$ , then it rejects in  $\mathcal{A}$  as well. Otherwise, it carries on its decision process by interpreting its certificate as a non-negative integer  $k$ , and accepts in  $\mathcal{A}$  unless  $f_M^{(k)}(a) = 1$  and  $f_M^{(k)}(b) = 1$ . To show the correctness of  $\mathcal{A}$ , let  $(G, x) \in \text{ITER}$ . We have  $f_{|L|,L} = 0$  or  $f_{|R|,R} = 0$ , i.e.,  $f_M^{(|L|)}(a) = 0$  or  $f_M^{(|R|)}(b) = 0$ . W.l.o.g., assume  $f_M^{(|L|)}(a) = 0$ . If  $k \geq |L|$  then  $f_M^{(k)}(a) = 0$  since  $f_M(0) = 0$ , and thus  $v$  accepts. If  $k < |L|$  then  $f_M^{(k)}(a) \neq 1$  since  $f_M(1) = 1$ , and thus  $v$  accepts. Therefore, all certificates lead to acceptance. Let us now consider  $(G, x) \notin \text{ITER}$ . If  $(G, x) \notin \text{ITER}^-$  then at least one node rejects, independently of the certificate. So, we assume that  $(G, x) \in \text{ITER}^- \setminus \text{ITER}$ . Thus,  $f_M^{(|L|)}(a) = 1$  and  $f_M^{(|R|)}(b) = 1$ . The certificate is set to  $k = \max\{|L|, |R|\}$ . Let us assume, w.l.o.g., that  $k = |L| \geq |R|$ . By this setting, we have  $f_M^{(k)}(a) = 1$ . Moreover, since  $k \geq |R|$ , and since  $f_M(1) = 1$ , we get that  $f_M^{(k)}(b) = 1$ . Therefore,  $\mathcal{A}$  rejects, as desired. Thus,  $\text{ITER} \in \Pi_1^{\text{loc}}$ .

It remains to show that  $\text{ITER} \notin \text{LD}$ . Let us assume, for the purpose of contradiction, that there exists a  $t$ -round algorithm  $\mathcal{A}$  deciding  $\text{ITER}$ . Since  $\text{ITER}^- \in \text{LD}$ , this algorithm is able to distinguish an instance with  $f_M^{(|L|)}(a) = 1$  and  $f_M^{(|R|)}(b) = 1$  from instances in which  $f_M^{(|L|)}(a) \neq 1$  or  $f_M^{(|R|)}(b) \neq 1$ . Observe that a node at distance greater than  $t$  from

## 8:10 What Can Be Verified Locally?

the pivot can gather information related to only one of the two inputs  $a$  and  $b$ . Therefore, the distinction between the case  $f_M^{(L)}(a) = 1$  and  $f_M^{(R)}(b) = 1$  and the case  $f_M^{(L)}(a) \neq 1$  or  $f_M^{(R)}(b) \neq 1$  can only be made by a node at distance at most  $t$  from the pivot. Therefore, by simulating  $\mathcal{A}$  at all nodes in the ball of radius  $t$  around  $v$ , with identities between 1 and the size of the ball of radius  $2t$  around the pivot, a sequential algorithm can determine, given a Turing machine  $M$ , and given  $a$  and  $b$ , whether there exist  $\ell$  and  $r$  such that  $f_M^{(\ell)}(a) = f_M^{(r)}(b) = 1$  or not, which is actually Turing undecidable. This contradiction implies that, indeed,  $\text{ITER} \notin \text{LD}$ . ◀

### 4 Complement classes

Given a class  $\mathcal{C}$  of distributed languages, the class  $\text{co-}\mathcal{C}$  is composed of all distributed languages  $\mathcal{L}$  such that  $\bar{\mathcal{L}} \in \mathcal{C}$ , where  $\bar{\mathcal{L}} = \{(G, x) \notin \mathcal{L}\}$ . For instance,  $\text{co-}\Pi_1^{\text{loc}}$  is the class of languages  $\mathcal{L}$  for which there exists a local algorithm  $\mathcal{A}$  such that, for every configuration  $(G, x)$ ,

$$\begin{aligned} (G, x) \in \mathcal{L} &\Rightarrow \exists c, \forall id, \exists u \in V(G), \mathcal{A}_{G, x, c, id}(u) = \text{accepts}; \\ (G, x) \notin \mathcal{L} &\Rightarrow \forall c, \forall id, \forall u \in V(G), \mathcal{A}_{G, x, c, id}(u) = \text{rejects}. \end{aligned}$$

Note in particular, that the rejection must now be unanimous, while the acceptance requires only one node to accept. Let us define the following two languages: each input to every node belongs to  $\{\text{true}, \text{false}\} = \{1, 0\}$ , and a configuration is in **AND** (resp., in **OR**) if and only if the logical conjunction (resp., disjunction) of the inputs is true. These two languages enable to separate LD from its co-class. Indeed,  $\text{OR} \notin \text{LD}$  as every node that sees only zeros must accept because there might exist far away nodes with input 1. Hence, an all-0 instance would be accepted, which is incorrect. Instead,  $\text{AND} \in \text{LD}$ : every node accepts if and only if its input is 1. The class  $\text{LD} \cap \text{co-LD}$  is quite restricted. Nevertheless, it contains distributed languages such as  $\text{DIAM}_k$ , the class of graphs with diameter at most  $k$ , for any fixed  $k$ . We have the following separation.

► **Proposition 8.**  $\text{OR} \in \text{co-LD} \setminus \Pi_1^{\text{loc}}$ , and  $\text{AND} \in \text{LD} \setminus \text{co-}\Pi_1^{\text{loc}}$ .

Similarly, the languages **ALTS** and **AMOS** introduced in the proof of Proposition 6 enable to separate **NLD** from its co-class. Indeed,  $\text{ALTS} = \overline{\text{AMOS}}$ , **ALTS** is closed under lift, and **AMOS** is not closed under lift. Moreover, consider the language **EXTS** defined at the end of Section 2. Both **EXTS** and  $\overline{\text{EXTS}}$  are not closed under lift. So, overall, by Lemma 4, we get:

► **Proposition 9.**  $\text{ALTS} \in \text{NLD} \setminus \text{co-NLD}$ ,  $\text{AMOS} \in \text{co-NLD} \setminus \text{NLD}$ , and  $\text{EXTS} \notin \text{NLD} \cup \text{co-NLD}$ .

More interesting is the position of the  $\Pi_1^{\text{loc}}$  w.r.t. **NLD** and **co-NLD**:

► **Proposition 10.**  $\Pi_1^{\text{loc}} \cup \text{co-}\Pi_1^{\text{loc}} \subset \text{NLD} \cap \text{co-NLD}$ , where the inclusion is strict.

### 5 Complete problems

In this section, we prove Theorem 2. Let  $G$  be a connected graph, and  $U$  be a set (typically,  $U = \{0, 1\}^*$ ). Let  $e : V(G) \rightarrow U$ , and let  $\mathcal{S} : V(G) \rightarrow 2^{2^U}$ . That is,  $e$  assigns an element  $e(u) \in U$  to every node  $u \in V(G)$ , and  $\mathcal{S}$  assigns a collection of sets  $\mathcal{S}(u) = \{S_1(u), \dots, S_{k_u}(u)\}$  to every node  $u \in V(G)$ , with  $k_u \geq 1$  and  $S_i : V(G) \rightarrow 2^U$  for every  $i \geq 1$ . We say that  $\mathcal{S}$  covers  $e$  if and only if there exists  $u \in V(G)$ , and there exists  $i \in \{1, \dots, k_u\}$ , such that  $S_i(u) = \{e(v) \mid v \in V(G)\}$ . In [6], the authors defined the language

$$\text{COVER} = \{(G, x) : \forall u \in V(G), x(u) = (\mathcal{S}(u), e(u)) \text{ such that } \mathcal{S} \text{ covers } e\}$$



and proved that COVER is the “most difficult decision task”, in the sense that every distributed language can be locally reduced to COVER. However COVER is closed under lift as lifting does not create new elements and preserves the sets. Therefore, by Lemma 4,  $\text{COVER} \in \text{NLD}$ .<sup>1</sup> This is in contradiction with the claim in [6] regarding the hardness of COVER. The reason for this contradiction is that the local reduction used in [6] for reducing any language to COVER is too strong. Indeed, it transforms a configuration  $(G, x)$  into a configuration  $(G, x')$  where the certificates used for proving  $x'$  may depend on the identities of the nodes in  $G$ . This is in contradiction with the definitions of the classes  $\Sigma_k^{\text{loc}}$  and  $\Pi_k^{\text{loc}}$ ,  $k \geq 0$ , for which the certificates must be independent of the identity assignment. In this section, we show that completeness results can be obtained using a more constrained notion of reduction which preserves the membership to the classes.

Recall from [6] that a local reduction of  $\mathcal{L}$  to  $\mathcal{L}'$  is a local algorithm  $\mathcal{R}$  which maps any configuration  $(G, x)$  to a configuration  $(G, y)$ , where  $y = R(G, x, \text{id})$  may depend on the identity assignment  $\text{id}$ , such that:  $(G, x) \in \mathcal{L}$  if and only if, for every identity assignment  $\text{id}$  to the nodes of  $G$ ,  $(G, y) \in \mathcal{L}'$  where  $y = \mathcal{R}(G, x, \text{id})$ . Ideally, we would like  $\mathcal{R}$  to be *identity-oblivious*, that is, such that the output of each node does not depend on the identity assignment, but this appears to be too restrictive. So, instead, we use a concept somewhat intermediate between identity-oblivious reduction and the unconstrained reduction in [6].

► **Definition 11.** Let  $\mathcal{C}$  be a class of distributed languages, and let  $\mathcal{L}$  and  $\mathcal{L}'$  be two distributed languages. Let  $\mathcal{A}$  be a  $\mathcal{C}$ -algorithm deciding  $\mathcal{L}'$ , and let  $\mathcal{R}$  be a local reduction of  $\mathcal{L}$  to  $\mathcal{L}'$ . We say that  $(\mathcal{R}, \mathcal{A})$  is *label-preserving* for  $(\mathcal{L}, \mathcal{L}')$  if and only if, for any configuration  $(G, x)$ , the existential certificates used by the prover in  $\mathcal{A}$  for  $(G, y)$  where  $y = \mathcal{R}(G, x, \text{id})$  are the same for all identity assignments  $\text{id}$  to  $G$ .

The following result shows that the notion of reduction in Definition 11 preserves the classes of distributed languages.

► **Lemma 12.** *Let  $\mathcal{C}$  be a class of distributed languages. Let  $\mathcal{L}$  and  $\mathcal{L}'$  be two distributed languages with  $\mathcal{L}' \in \mathcal{C}$ , and let  $(\mathcal{R}, \mathcal{A})$  be a label-preserving local reduction for  $(\mathcal{L}, \mathcal{L}')$ . Then  $\mathcal{L} \in \mathcal{C}$ .*

We now exhibit a language that is among the hardest decision tasks, under local label-preserving reductions. In the following decision task, every node  $u$  of a configuration  $(G, x)$  is given a family  $\mathcal{F}(u)$  of configurations, each described by an adjacency matrix representing a graph, and a 1-dimensional array representing the inputs to the nodes of that graph. In addition, every node  $u$  has an input string  $x'(u) \in \{0, 1\}^*$ . Hence,  $(G, x')$  is also a configuration. The actual configuration  $(G, x)$  is legal if  $(G, x')$  is missing in all families  $\mathcal{F}(u)$  for every  $u \in V(G)$ , i.e.,  $(G, x') \notin \mathcal{F}$  where  $\mathcal{F} = \cup_{u \in V(G)} \mathcal{F}(u)$ . In short, we consider the language

$$\text{MISS} = \{(G, x) : \forall u \in V(G), x(u) = (\mathcal{F}(u), x'(u)) \text{ and } (G, x') \notin \mathcal{F}\}$$

We show that MISS is among the hardest decision tasks, under local label-preserving reductions. Note that  $\text{MISS} \notin \text{NLD}$  (it is not closed under lift: it may be the case that  $(G, x') \notin \mathcal{F}$  but a lift of  $(G, x')$  is in  $\mathcal{F}$ ).

► **Proposition 13.** *MISS is  $\Pi_2^{\text{loc}}$ -complete under local label-preserving reductions.*

<sup>1</sup> In fact, one can show that there exists a local verification algorithm for COVER using certificates of size quasi linear in  $n$  whenever the ground set  $U$  is of polynomial size.

**Proof.** Let  $\mathcal{L}$  be a distributed language. We describe a local label-preserving reduction  $(R, \mathcal{A})$  for  $(\mathcal{L}, \text{MISS})$  with respect to  $\Pi_2^{\text{loc}}$ .

In essence, the local algorithm  $\mathcal{A}$  for deciding MISS in  $\Pi_2^{\text{loc}}$  is the generic algorithm described in the proof of Proposition 3. Recall that, in this generic algorithm, on a legal configuration  $(G, x)$ , the existential  $c_2$  certificate in  $\mathcal{A}$  is pointing to an inconsistency in the given  $c_1$  certificate which is supposed to describe the configuration  $(G, x)$ . And, on an illegal configuration  $(G, x)$ , the existential  $c_1$  certificate in  $\mathcal{A}$  does provide an accurate description of the configuration  $(G, x)$ . For the purpose of label-preservation, we slightly modify the generic algorithm for MISS. Instead of viewing  $c_1$  as a description of the configuration  $(G, x)$ , the algorithm views it as a description of  $(G, x')$  where, at each node  $u$ ,  $x'(u)$  is the second item in  $x(u)$  (the first item is the family  $\mathcal{F}(u)$ ). The algorithm is then exactly the same as the generic algorithm with the only modification that the test when the flag  $f(u) = 1$  is not regarding whether  $(G, x') \in \text{MISS}$ , but whether  $(G, x') \notin \mathcal{F}(u)$ . On a legal configuration, all nodes accept. On an illegal instance, a node with  $(G, x') \in \mathcal{F}(u)$  rejects.

The reduction  $R$  from  $\mathcal{L}$  to MISS proceeds as follows, in a way similar to the one in [6]. A node  $u$  with identity  $\text{id}(u)$  and input  $x(u)$  computes its *width*  $\omega(u) = 2^{|\text{id}(u)| + |x(u)|}$  where  $|s|$  denotes the length of a bit-string  $s$ . Then  $u$  generates all configurations  $(H, y) \notin \mathcal{L}$  such that  $H$  has at most  $\omega(u)$  nodes and  $y(v)$  has value at most  $\omega(u)$ , for every node  $v$  of  $H$ . It places all these configurations in  $\mathcal{F}(u)$ . The input  $x'(u)$  is simply  $x'(u) = x(u)$ . If  $(G, x) \in \mathcal{L}$ , then  $(G, x) \notin \mathcal{F}$  since only illegal instances are in  $\mathcal{F}$ , and thus  $(G, R(G, x)) \in \text{MISS}$ . Conversely, if  $(G, x) \notin \mathcal{L}$ , then  $(G, R(G, x)) \notin \text{MISS}$ . Indeed, there exists at least one node  $u$  with identity  $\text{id}(u) \geq n$ , which guarantees that  $u$  generates the graph  $G$ . If no other node  $u'$  has width  $\omega(u') > n$  then  $u$  generates  $(G, x) \in \mathcal{F}(u)$ . If there exists a node  $u'$  with  $\omega(u') > n$  then  $u'$  generates  $(G, x) \in \mathcal{F}(u')$ . In each case, we have  $(G, x) \in \mathcal{F}$ , and thus  $(G, R(G, x)) \notin \text{MISS}$ .

It remains to show that the existential certificate used in  $\mathcal{A}$  for all configurations  $(G, R(G, x))$  are the same for any given  $(G, x)$ , independently of the identity assignment to  $G$  used to perform the reduction  $R$ . This directly follows from the nature of  $\mathcal{A}$  since the certificates do not depend on the families  $\mathcal{F}(u)$ 's but only on the bit strings  $x'(u)$ 's. ◀

The following language is defined as MISS by replacing  $\mathcal{F}$  by the closure under lift  $\mathcal{F}^\uparrow$  of  $\mathcal{F}$ . That is,  $\mathcal{F}^\uparrow$  is composed of  $\mathcal{F}$  and all the lifts of the configurations in  $\mathcal{F}$ .

$$\text{MISS}^\uparrow = \{(G, x) : \forall u \in V(G), x(u) = (\mathcal{F}(u), x'(u)) \text{ and } (G, x') \notin \mathcal{F}^\uparrow\}$$

We show that  $\text{MISS}^\uparrow$  is among the hardest decision tasks in NLD.

► **Proposition 14.**  *$\text{MISS}^\uparrow$  is NLD-complete (and  $\overline{\text{MISS}^\uparrow}$  is co-NLD-complete) under label-preserving reduction.*

## 6 Conclusion

This paper is aiming at providing a proof of concept for the notion of interactive local verification:  $\Pi_2^{\text{loc}}$  can be viewed as the interaction between two players, with conflicting objectives, one is aiming at proving the instance, while the other is aiming at disproving it. As a consequence, for this first attempt, we voluntarily ignored important parameters such as the size of the certificates, and the individual computation time, and we focussed only on the locality issue. The impact of limiting the certificate size was recently investigated in [2]. Regarding the individual computation time, our completeness results involve local reductions that are very much time consuming at each node. Insisting on local reductions involving polynomial-time computation at each node is crucial for practical purpose. At this point, we

do not know whether non-trivial hardness results can be established under polynomial-time local reductions. Proving or disproving the existence of such hardness results is left as an open problem.

**Acknowledgement.** The authors are thankful to Laurent Feuilloley for fruitful discussions about the topic of the paper.

---

## References

---

- 1 L. Feuilloley, P. Fraigniaud: Randomized Local Network Computing. In Proc. 27th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), pp. 340–349, 2015
- 2 L. Feuilloley, P. Fraigniaud, J. Hirvonen: A Hierarchy of Local Decision. In Proc. 43rd International Colloquium on Automata, Languages and Programming (ICALP), pp. 118:1–118:15, 2016. doi:10.4230/LIPIcs.ICALP.2016.118
- 3 P. Floréen, J. Kaasinen, P. Kaski, J. Suomela: An optimal local approximation algorithm for max-min linear programs. In Proc. 21st ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), pp. 260–269, 2009.
- 4 P. Fraigniaud, M. Göös, A. Korman, J. Suomela: What can be decided locally without identifiers? In Proc. 32nd ACM Symp. on Principles of Distributed Computing (PODC), pp. 157–165, 2013.
- 5 P. Fraigniaud, M. Halldórsson, A. Korman. On the Impact of Identifiers on Local Decision. In Proc. 16th Int. Conference on Principles of Distributed Systems (OPODIS). Springer, LNCS 7702, pp. 224–238, 2012.
- 6 P. Fraigniaud, A. Korman, D. Peleg. Towards a complexity theory for local distributed computing. J. ACM 60(5): 35 (2013) (Preliminary version in FOCS 2011).
- 7 M. Göös, J. Suomela: Locally checkable proofs. In Proc. 30th ACM Symposium on Principles of Distributed Computing (PODC), pp. 159–168, 2011.
- 8 A. Korman, S. Kutten, D. Peleg (2010). Proof labeling schemes. Distributed Computing 22(4):215–233.
- 9 F. Kuhn, T. Moscibroda, R. Wattenhofer: What cannot be computed locally! In Proc. 23rd ACM Symp. on Principles of Distributed Computing (PODC), pp. 300–309, 2004.
- 10 C. Lenzen, Y. Anne Oswald, R. Wattenhofer: What can be approximated locally? In Proc. 20th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), pp. 46–54, 2008.
- 11 C. Lenzen, R. Wattenhofer: Leveraging Linial's Locality Limit. In Proc. 22nd Int. Symp. on Distributed Computing (DISC), pp. 394–407, 2008.
- 12 N. Linial. Locality in Distributed Graph Algorithms. SIAM J. Comp. 21(1): 193-201 (1992)
- 13 M. Naor, L. Stockmeyer. What Can be Computed Locally? SIAM J. Comput. 24(6):1259–1277 (1995)
- 14 D. Peleg. Distributed Computing: A Locality-Sensitive Approach. SIAM (2000)
- 15 F. Reiter: Distributed Graph Automata. In Proc. 30th ACM/IEEE Symposium on Logic in Computer Science (LICS), pp. 192–201, 2015.
- 16 J. Suomela: Survey of local algorithms. ACM Comput. Surv. 45(2):24 (2013)



# Improved Time-Space Trade-Offs for Computing Voronoi Diagrams\*

Bahareh Banyassady<sup>1</sup>, Matias Korman<sup>2</sup>, Wolfgang Mulzer<sup>3</sup>,  
André van Renssen<sup>4</sup>, Marcel Roeloffzen<sup>5</sup>, Paul Seiferth<sup>6</sup>, and  
Yannik Stein<sup>7</sup>

- 1 Institut für Informatik, Freie Universität Berlin, Berlin, Germany  
bahareh@inf.fu-berlin.de
- 2 Tohoku University, Sendai, Japan  
mati@dais.is.tohoku.ac.jp
- 3 Institut für Informatik, Freie Universität Berlin, Berlin, Germany  
mulzer@inf.fu-berlin.de
- 4 National Institute of Informatics (NII), Tokyo, Japan; and  
JST, ERATO, Kawarabayashi Large Graph Project, Tokyo, Japan  
andre@nii.ac.jp
- 5 National Institute of Informatics (NII), Tokyo, Japan; and  
JST, ERATO, Kawarabayashi Large Graph Project, Tokyo, Japan  
marcel@nii.ac.jp
- 6 Institut für Informatik, Freie Universität Berlin, Berlin, Germany  
pseiferth@inf.fu-berlin.de
- 7 Institut für Informatik, Freie Universität Berlin, Berlin, Germany  
yannikstein@inf.fu-berlin.de

---

## Abstract

Let  $P$  be a planar  $n$ -point set in general position. For  $k \in \{1, \dots, n-1\}$ , the Voronoi diagram of order  $k$  is obtained by subdividing the plane into *regions* such that points in the same cell have the same set of nearest  $k$  neighbors in  $P$ . The (*nearest point*) Voronoi diagram (NVD) and the (*farthest point*) Voronoi diagram (FVD) are the particular cases of  $k = 1$  and  $k = n-1$ , respectively. It is known that the family of all higher-order Voronoi diagrams of order 1 to  $K$  for  $P$  can be computed in total time  $O(nK^2 + n \log n)$  using  $O(K^2(n-K))$  space. Also NVD and FVD can be computed in  $O(n \log n)$  time using  $O(n)$  space.

For  $s \in \{1, \dots, n\}$ , an  $s$ -workspace algorithm has random access to a read-only array with the sites of  $P$  in arbitrary order. Additionally, the algorithm may use  $O(s)$  words of  $\Theta(\log n)$  bits each for reading and writing intermediate data. The output can be written only once and cannot be accessed afterwards.

We describe a deterministic  $s$ -workspace algorithm for computing an NVD and also an FVD for  $P$  that runs in  $O((n^2/s) \log s)$  time. Moreover, we generalize our  $s$ -workspace algorithm for computing the family of all higher-order Voronoi diagrams of  $P$  up to order  $K \in O(\sqrt{s})$  in total time  $O\left(\frac{n^2 K^6}{s} \log^{1+\varepsilon} K \cdot (\log s / \log K)^{O(1)}\right)$ , for any fixed  $\varepsilon > 0$ . Previously, for Voronoi diagrams, the only known  $s$ -workspace algorithm was to find an NVD for  $P$  in *expected* time  $O((n^2/s) \log s + n \log s \log^* s)$ . Unlike the previous algorithm, our new method is very simple and does not rely on advanced data structures or random sampling techniques.

**1998 ACM Subject Classification** F.2.2 [Nonnumerical Algorithms and Problems] Geometrical Problems and Computations

---

\* MK was supported in part by the ELC project (MEXT KAKENHI No. 12H00855 and 15H02665). BB, WM and PS were supported in part by DFG Grants MU 3501/1 and MU 3501/2. YS was supported by the DFG within the research training group “Methods for Discrete Structures” (GRK 1408) and by GIF Grant 1161.



**Keywords and phrases** memory-constrained model, Voronoi diagram, time-space trade-off

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.9

## 1 Introduction

In recent years, we have seen an explosive growth of small distributed devices such as tracking devices and wireless sensors. These devices are small, have only limited energy supply, are easily moved, and should not be too expensive. To accommodate these needs, the amount of memory on them is tightly budgeted. This poses a significant challenge to software developers and algorithm designers: how to create useful and efficient programs in the presence of strong memory constraints?

Memory constraints have been studied since the introduction of computers (see for example Pohl [27]). The first computers often had limited memory compared to the available processing power. As hardware progressed this gap became smaller, other concerns became more important, and the focus of algorithms research shifted away from memory-constrained models. Memory constraints are again an important problem to tackle with these new devices as well as huge datasets available through cloud computing.

An easy way to model algorithms with memory constraints is to assume that the input is stored in a read-only memory. This is appealing for several reasons. From a practical viewpoint, writing to external memory is often a costly operation, e.g., if the data resides on a read-only medium such as a DVD or on hardware where writing is slow and wears out the hardware, such as flash memory. Similarly, in concurrent environments, writing operations may lead to race conditions. Thus, from a practical viewpoint, it is useful to limit or simply disallow writing operations. From a theoretical viewpoint, this model is also advantageous: keeping the working memory separate from the (read-only) input memory allows for a more detailed accounting of the space requirements of an algorithm and for a better understanding of the required resources. In fact, this is exactly the approach taken by computational complexity theory to define complexity classes that model *sublinear* space requirements, such as the complexity class of problems that use logarithmic amount of memory space [3].

Some of the earliest results in this setting concern the sorting problem [24, 25]. Suppose we want to sort data items whose total description complexity is  $n$  bits, all of them residing in a read-only memory. For our computations we can use a workspace of  $O(b)$  bits freely (both read and write operations are allowed). Then it is known that the time-space product must be  $\Omega(n^2)$  [13], and a matching upper bound for the case  $b \in \Omega(\log n) \cap O(n/\log n)$  was given by Pagter and Rauhe [26] ( $b$  is the available workspace in bits). A result along these lines is known as a *time-space trade-off* [28].

The model used in this work was introduced by Asano *et al.* [6], following similar earlier models [14, 17]. Asano *et al.* provided constant workspace algorithms for many classic problems from computational geometry, such as computing convex hulls, Delaunay triangulations, Euclidean minimum spanning trees, or shortest paths in polygons [6]. Since then, the model has enjoyed increasing popularity, with work on shortest paths in trees [7] and time-space trade-offs for computing shortest paths [4, 20], visibility regions in simple polygons [9, 11], planar convex hulls [10, 18], general plane-sweep algorithms [19], or triangulating simple polygons [4, 5, 2]. We refer the reader to [21] for a deeper survey of the latest results.

Let us specify our model more precisely: we are given a planar point set  $P$  of  $n$  points stored in a read-only array that allows random access. Furthermore, we may use  $O(s)$  variables (for a parameter  $s \in \{1, \dots, n\}$ ) for reading and writing. We assume that all the

data items and pointers are represented by  $\Theta(\log n)$  bits. Other than this, the model allows the usual word RAM operations.

We consider the problem of computing various Voronoi diagrams for  $P$ , namely the *nearest point Voronoi diagram*  $\text{NVD}(P)$ , the *furthest point Voronoi diagram*  $\text{FVD}(P)$ , and the family of *higher-order Voronoi diagrams* up to a given order  $K \in O(\sqrt{s})$ . In most workspaces, the output cannot be stored explicitly. Thus, we require that the algorithm reports the edges of the Voronoi diagrams one-by-one in a write-only data structure (once written, they cannot be read or further modified). Note that we may report edges of the Voronoi diagrams in any order, but we are not allowed to report an edge more than once.

**Previous Work.** If we forego memory constraints, it is well known that both  $\text{NVD}(P)$  and  $\text{FVD}(P)$  can be computed in  $O(n \log n)$  time using  $O(n)$  space [8, 12]. For computing a single order- $k$  Voronoi diagram, the best randomized algorithm takes  $O(n \log n + nk \log k)$  expected time [16] using  $O(nk)$  space, and the best deterministic algorithm takes  $O(nk \log^{1+\varepsilon} k \cdot (\log n / \log k)^{O(1)})$  time [15] and  $O(nk)$  space, for  $\varepsilon > 0$  (as usual, the big  $O$  notation hides multiplicative factors that depend on  $\varepsilon$ ). The family of all higher-order Voronoi diagrams up to order  $K$  can be computed in  $O(nK^2 + n \log n)$  time using  $O(K^2(n - K))$  space [1, 23].

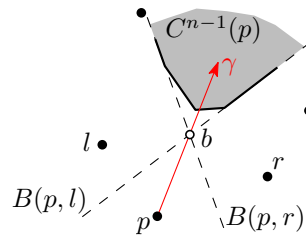
Very few memory-constrained algorithms that compute Voronoi diagrams exist in the literature. Asano *et al.* [6] showed that  $\text{NVD}(P)$  can be found in  $O(n^2)$  time in a  $O(1)$  workspace. Korman *et al.* [22] gave a time-space trade-off for computing Voronoi diagrams. Their algorithm is based on random sampling and achieves an expected running time of  $O((n^2/s) \log s + n \log s \log^* s)$  using  $O(s)$  words of workspace.

**Results.** In this paper we introduce a time-space trade-off algorithm that improves these results, and gives a simpler and more flexible approach to obtain the diagrams. In Section 3 we show that the approach of Asano *et al.* [6] can be used to compute  $\text{FVD}(P)$ . In Section 4 we introduce a new time-space trade-off for computing  $\text{NVD}(P)$  and  $\text{FVD}(P)$ . Unlike the approach of Korman *et al.* [22], this new algorithm is deterministic, and slightly faster (it runs in  $O((n^2/s) \log s)$  time using  $O(s)$  words of workspace).

Finally, in Section 5, we use the  $s$ -workspace algorithm as a base in a novel pipelined fashion to compute the family of all Voronoi diagrams of order 1 to  $K \in O(\sqrt{s})$  in total time  $O(\frac{n^2 K^6}{s} \log^{1+\varepsilon} K \cdot (\log s / \log K)^{O(1)})$ , for  $\varepsilon > 0$ , using  $O(s)$  words of workspace. The main idea is to compute edges of the different Voronoi diagrams simultaneously. To compute the edges of a diagram we use edges of the previous order Voronoi diagram. However, this needs to be coordinated carefully, in order to prevent edges from being reported multiple times.

## 2 Preliminaries and Notation

Throughout the paper we denote by  $P = \{p_1, \dots, p_n\}$  a set of  $n \geq 3$  sites in the plane. We assume general position, which here means that no three sites of  $P$  lie on a common line and no four sites of  $P$  lie on a common circle. The *nearest point Voronoi diagram* of  $P$ ,  $\text{NVD}(P)$ , is obtained by classifying the points in the plane according to their nearest neighbors in  $P$ . To define our terminology, we recall some classic and well-known properties of  $\text{NVD}(P)$  [8, 12]. For each site  $p \in P$ , the open set of points in  $\mathbb{R}^2$  that have  $p$  as their unique nearest neighbor in  $P$  is called the *Voronoi cell* of  $p$ . Each *Voronoi edge* between two points  $p, q \in P$  consists of all points in the plane with  $p, q$  as their only two nearest neighbors. Whenever it exists, the Voronoi edge is a subset of the bisector  $B(p, q)$  of  $p, q$  defined as the line containing all points that are equidistant to  $p$  and  $q$ . Note that our general position



■ **Figure 1** An illustration of Fact 3.2: if  $p$  is on  $\text{conv}(P)$ , we can find a ray that intersects the boundary of  $C^{n-1}(p)$ .

assumption together with  $n \geq 3$  guarantees that each Voronoi edge is an open segment or halfline. Finally, *Voronoi vertices* are the points in the plane that have exactly three nearest neighbors in  $P$ . By general position, every point in  $\mathbb{R}^2$  is either a Voronoi vertex, or it lies on a Voronoi edge or in a Voronoi cell. The Voronoi vertices and the Voronoi edges form the set of vertices and edges of a plane graph whose faces are the Voronoi cells. The complexity of this graph is  $O(n)$ .

The *farthest point Voronoi diagram* of  $P$ ,  $\text{FVD}(P)$ , is defined analogously. Farthest Voronoi cells, edges, and vertices are obtained by replacing the term “nearest neighbor” by the term “farthest neighbor” in the respective definitions. Again, the farthest Voronoi vertices and edges constitute the vertices and edges of a plane graph of complexity  $O(n)$ . However, unlike in  $\text{NVD}(P)$ , in  $\text{FVD}(P)$  it is not necessarily the case that all sites in  $P$  have a corresponding cell in  $\text{FVD}(P)$ . In fact, the sites with non-empty farthest Voronoi cells correspond exactly to the sites on the *convex hull* of  $P$ ,  $\text{conv}(P)$ . Furthermore, all the cells in  $\text{FVD}(P)$  are unbounded, and hence  $\text{FVD}(P)$ , considered as a plane graph, is a tree.

Now for  $k \in \{1, \dots, n-1\}$ , the Voronoi diagram of order  $k$  is obtained by classifying the points in the plane according to the *set* of their nearest  $k$  neighbors in  $P$ . We denote the  $k$ -order Voronoi diagram of  $P$  by  $\text{VD}^k(P)$ . Observe that  $\text{NVD}(P) = \text{VD}^1(P)$  and  $\text{FVD}(P) = \text{VD}^{n-1}(P)$ . For each subset  $Q \subset P$  of  $k$  sites from  $P$ , we denote the Voronoi cell of order  $k$  of  $Q$  by  $C^k(Q)$ . We know that  $\text{VD}^k(P)$  is a plane graph of complexity  $O(k(n-k))$  [8, 23]. For simplicity, the Voronoi cell of  $p \in P$  in  $\text{NVD}(P)$  and  $\text{FVD}(P)$  are denoted respectively by  $C^1(p)$  and  $C^{n-1}(p)$ .

### 3 A Constant Workspace Algorithm for FVDs and NVDs

We are given a planar  $n$ -point set  $P = \{p_1, \dots, p_n\}$  in a read-only array, and our task is to report the edges of  $\text{NVD}(P)$  and of  $\text{FVD}(P)$  using only a constant amount of additional workspace. First, we show how to find a single edge of a cell of  $\text{NVD}(P)$  or of  $\text{FVD}(P)$ . Then, we extend this approach to find all the edges of  $\text{NVD}(P)$  and  $\text{FVD}(P)$ . We summarize the properties of  $\text{FVD}(P)$ , that are required by our algorithms, in the following two facts. See, e.g., the book by Aurenhammer, Klein, and Lee [8] for more details.

► **Fact 3.1.** *Let  $P$  be a planar  $n$ -point set in general position and  $p \in P$ . The cell  $C^{n-1}(p)$  is empty if and only if  $p$  is in the interior of the convex hull of  $P$ . If  $p$  is on the convex hull of  $P$  and  $r, l \in P$  are the two adjacent sites of  $p$  on  $\text{conv}(P)$ , then both a subset of  $B(p, l)$  and a subset of  $B(p, r)$  are unbounded edges of  $C^{n-1}(p)$ .*

► **Fact 3.2.** *Let  $P$  be a planar  $n$ -point set in general position. Let  $p \in P$  be on  $\text{conv}(P)$  and let  $l, r \in P$  be its adjacent sites on  $\text{conv}(P)$ . Let  $b$  be the intersection point of  $B(p, l)$  and*



$B(p, r)$ . Then, the ray  $\gamma$  from  $p$  towards  $b$  intersects the boundary of  $C^{n-1}(p)$  (not necessarily at  $b$ ); see Figure 1.

► **Lemma 3.3.** *Let  $P$  be a planar  $n$ -point set in general position in a read-only array. For any  $p \in P$ , we can determine whether  $C^{n-1}(p)$  is empty, in  $O(n)$  time and constant workspace. Furthermore, if  $C^{n-1}(p)$  is not empty, we can find a ray that intersects the boundary of  $C^{n-1}(p)$  in the same time and space.*

**Proof.** By Fact 3.1, it suffices to check whether  $p$  lies inside  $\text{conv}(P)$ . This can be done using simple *gift-wrapping*: Pick an arbitrary site  $q \in P \setminus \{p\}$ . Scan through  $P$  and find the sites  $p_{\text{cw}}$  and  $p_{\text{ccw}}$  in  $P$  which make, respectively, the largest clockwise angle and the largest counterclockwise angle with the ray  $pq$ , such that both angles are at most  $\pi$ . Thus,  $p_{\text{cw}}$  and  $p_{\text{ccw}}$  are easily obtained in  $O(n)$  time using constant workspace. If the angle  $p_{\text{cw}}pp_{\text{ccw}}$  (that contains  $q$ ) is larger than  $\pi$ , then  $p$  is inside  $\text{conv}(P)$  and consequently  $C^{n-1}(p)$  is empty. Otherwise,  $p$  is on  $\text{conv}(P)$ , and both  $p_{\text{cw}}$  and  $p_{\text{ccw}}$  are its adjacent sites on  $\text{conv}(P)$ . By Fact 3.2, the ray from  $p$  through  $B(p, p_{\text{cw}}) \cap B(p, p_{\text{ccw}})$  intersects the boundary of  $C^{n-1}(p)$ . ◀

► **Lemma 3.4.** *Let  $P$  be a planar  $n$ -point set in general position in a read-only array. Suppose we are given a site  $p \in P$  and a ray  $\gamma$  that emanates from  $p$  and intersects the boundary of  $C^1(p)$  (or  $C^{n-1}(p)$ ). Then, we can report the edge  $e$  of  $C^1(p)$  (or  $C^{n-1}(p)$ ) that intersects  $\gamma$ , in  $O(n)$  time using  $O(1)$  words of workspace.*

**Proof.** For all sites  $p' \in P$ , we consider the bisector  $B(p, p')$ . Among all these bisectors, we find the bisector  $\ell_e$  that intersects  $\gamma$  closest to (farthest from)  $p$ . The edge  $e$  is a subset of  $\ell_e$ . We can find  $\ell_e$  by scanning the sites of  $P$  and storing the closest (farthest) bisector so far in each step. To find the portion of  $\ell_e$  that forms a Voronoi edge in  $\text{NVD}(P)$  (or  $\text{FVD}(P)$ ), we do a second scan of  $P$ . For any  $p' \in P$  we check where  $B(p, p')$  intersects  $\ell_e$ . Each such intersection removes a section from  $\ell_e$  which cannot appear in  $\text{NVD}(P)$  (or  $\text{FVD}(P)$ ). From each infinite side of  $\ell_e$ , there is at most one intersection that removes the biggest portion of  $\ell_e$  and thus defines the endpoint of  $e$  from that side. Thus, in each step we store only the most restricted intersection from each side (if it exists). Overall, we can find the edge  $e$  of  $C^1(p)$  (or of  $C^{n-1}(p)$ ) in  $O(n)$  time using  $O(1)$  words of workspace. ◀

► **Theorem 3.5.** *Suppose we are given a planar  $n$ -point set  $P = \{p_1, \dots, p_n\}$  in general position in a read-only array. We can find all the edges of  $\text{NVD}(P)$  (or of  $\text{FVD}(P)$ ) in  $O(n^2)$  time using  $O(1)$  words of workspace.*

**Proof.** We restate the strategy that was previously used by Asano *et al.* [6] for  $\text{NVD}(P)$ . We give the details and show that a similar strategy works for  $\text{FVD}(P)$ .

We go through the sites in  $P$  one by one. In step  $i$ , we process  $p_i \in P$  to detect all edges of  $C^1(p_i)$  (or  $C^{n-1}(p_i)$ ). To do this, we first need a ray  $\gamma$  to apply Lemma 3.4. For  $\text{NVD}(P)$  we choose a ray  $\gamma$  from  $p_i$  to an arbitrary site of  $P \setminus \{p_i\}$ . In this way, we know that  $\gamma$  intersects the boundary of  $C^1(p_i)$ . For  $\text{FVD}(P)$  first check if the Voronoi cell of  $p_i$  is non-empty. If so, we use Lemma 3.3 to find a ray  $\gamma$  that intersects the boundary of  $C^{n-1}(p_i)$ . From here on, the algorithms for enumerating the edges of  $\text{NVD}(P)$  and  $\text{FVD}(P)$  are similar. Having the ray  $\gamma$  at hand, we use Lemma 3.4 to find an edge  $e$  of  $C^1(p_i)$  (or of  $C^{n-1}(p_i)$ ). We consider the ray  $\gamma'$  from  $p_i$  to the left endpoint of  $e$  (if it exists), and we apply Lemma 3.4 to find the adjacent edge  $e'$  of  $e$  in  $C^1(p_i)$  (or in  $C^{n-1}(p_i)$ ). Note that the ray will now hit both  $e$  and  $e'$ . This can be fixed by making a symbolic perturbation to  $\gamma'$  so that only  $e'$  is hit. We proceed in a similar manner to find further edges of  $C^1(p_i)$  (or  $C^{n-1}(p_i)$ ) in counterclockwise direction. The process continues until we reach  $e$  again or until we find an

unbounded edge of  $C^1(p_i)$  (or of  $C^{n-1}(p_i)$ ). In the latter case, we start again from the right endpoint of  $e$  (if it exists), and we find the remaining edges of  $C^1(p_i)$  (or of  $C^{n-1}(p_i)$ ) in clockwise direction, stopping the process when the current edge is unbounded.

Using this process we detect each edge twice (i.e., edges that are a subset of  $B(p_i, p_j)$  will be detected when processing  $p_i$  and  $p_j$ ). To avoid reporting the same edge twice, when we find an edge  $e$  of  $C^1(p_i)$  (or of  $C^{n-1}(p_i)$ ) with  $e \subseteq B(p_i, p_j)$  we report  $e$  if and only if  $i < j$ . Since  $\text{NVD}(P)$  (or  $\text{FVD}(P)$ ) has  $O(n)$  edges, and reporting one edge takes  $O(n)$  time and  $O(1)$  words of workspace, the result follows. ◀

#### 4 Obtaining a Time-Space Trade-off

Now we adapt the previous algorithm to a time-space trade-off in which we have a workspace of  $O(s)$  variables. As before, we are given a planar  $n$ -point set  $P = \{p_1, \dots, p_n\}$  in general position in a read-only array, and we would like to report the edges of  $\text{NVD}(P)$  or  $\text{FVD}(P)$  as quickly as possible. For this, we first show how to find one edge of  $s$  different cells of  $\text{NVD}(P)$  or  $\text{FVD}(P)$  simultaneously. After that, we describe how to coordinate these simultaneous searches to find all the edges of  $\text{NVD}(P)$  or  $\text{FVD}(P)$ .

► **Lemma 4.1.** *Suppose we are given a set  $V = \{v_1, \dots, v_s\}$  of  $s$  sites in  $P$ , and for each  $i = 1, \dots, s$ , a ray  $\gamma_i$  emanating from  $v_i$  such that  $\gamma_i$  intersects the boundary of  $C^1(v_i)$  (or  $\text{FVD}(P)$ ). Then we can report for each  $i = 1, \dots, s$ , the edge  $e_i$  of  $C^1(v_i)$  (or  $\text{FVD}(P)$ ) that intersects  $\gamma_i$ , in  $O(n \log s)$  total time using  $O(s)$  words of workspace.*

**Proof.** For ease of reading we provide the proof only for  $\text{NVD}(P)$  (the proof for  $\text{FVD}(P)$  is obtained by simply replacing  $\text{NVD}(P)$  by  $\text{FVD}(P)$ ). The algorithm has two phases. In the first phase, for  $i = 1, \dots, s$ , we find the line  $\ell_i$  that contains  $e_i$ , and in the second phase, for  $i = 1, \dots, s$ , we find the portion of  $\ell_i$  which is in  $\text{NVD}(P)$ , i.e., we find the endpoints of  $e_i$ .

The first phase proceeds as follows: we select the first batch  $Q_1 = \{p_1, \dots, p_s\}$ , of  $s$  sites of  $P$ , and we compute  $\text{NVD}(V \cup Q_1)$ . Since  $V \cup Q_1$  has  $O(s)$  sites, we can compute it in  $O(s \log s)$  time using  $O(s)$  workspace. Now, for  $i = 1, \dots, s$ , we find the edge  $e'_i \in \text{NVD}(V \cup Q_1)$  of the cell of  $v_i$  that intersects  $\gamma_i$ , we store the line spanned by  $e'_i$  in  $\ell_i$ , and proceed to the next batch of  $s$  sites. In general in step  $j$ , for  $j = 1, \dots, n/s$ , we select  $Q_j$  which is the  $j^{\text{th}}$  batch of  $s$  sites of  $P$ , and we compute  $\text{NVD}(V \cup Q_j)$ . Then, for  $i = 1, \dots, s$ , we find the edge of the cell of  $v_i$  in  $\text{NVD}(V \cup Q_j)$  that intersects  $\gamma_i$ . We update  $\ell_i$  to the line spanned by this new edge only if it intersects  $\gamma_i$  closest to  $v_i$ .

We claim that after all  $n/s$  batches of  $P$  have been scanned,  $\ell_i$  is the line that contains the edge of  $C^1(v_i)$  that intersects  $\gamma_i$ . To see this, recall that the edge  $e_i$  in  $\text{NVD}(P)$  lies on a bisector between  $v_i$  and another site  $p \in P \setminus \{v_i\}$ . Thus, this line is among the lines considered in  $\text{NVD}(V \cup Q_j)$ , for  $j = 1, \dots, n/s$ .

In the second phase, we again process  $P$  in batches of size  $s$ . In the first step, we take the first batch of  $s$  sites of  $P$ ,  $Q_1 = \{p_1, \dots, p_s\}$ , and we again compute  $\text{NVD}(V \cup Q_1)$ . For  $i = 1, \dots, s$ , we find the portion of  $\ell_i$  inside the cell of  $v_i$  in  $\text{NVD}(V \cup Q_1)$ , and we store it in  $e_i$ . In step  $j$ , for  $j = 1, \dots, n/s$ , we select  $Q_j$ , the  $j^{\text{th}}$  batch of  $s$  sites of  $P$ , and we compute  $\text{NVD}(V \cup Q_j)$ . For  $i = 1, \dots, s$ , we update the endpoints of  $e_i$  (the new  $e_i$  is simply the intersection of the previous  $e_i$  and the cell of  $v_i$  in  $\text{NVD}(V \cup Q_j)$ ). At the end of step  $j$ , the variable  $e_i$  represents the best candidate that we have found so far. In other words,  $e_i$  contains the portion of  $\ell_i$  whose nearest site is  $v_i$  (among the sites  $V \cup_{k=1}^j Q_k$ ). Further note that, due to the Voronoi properties,  $e_i$  is a connected subset of  $\ell_i$  (that is, a ray or a segment). In particular, it can be described with its at most two endpoints. Thus, after  $n/s$  steps,  $e_i$  is the edge of  $C^1(v_i)$  that intersects  $\gamma_i$ .

In each step of each phase, we construct a Voronoi diagram in  $O(s \log s)$  time using  $O(s)$  workspace. Since the total number of steps is  $n/s$ , the running time of the algorithm is  $O(n \log s)$ . At each step we store only  $O(s)$  sites (and a constant amount of information on each site), so the space bounds are not exceeded. ◀

Now we can describe our algorithm. We repeatedly use Lemma 4.1 to find an edge of  $s$  different sites at once. Once all edges of a site have been found, it is discarded and we proceed to the next one. Since the Voronoi diagram has  $O(n)$  edges and at each iteration we find  $s$  edges, after  $O(n/s)$  steps, fewer than  $s$  sites will remain to be processed. At this step we stop using Lemma 4.1. We do so because if the number of edges remaining to be found for each site is unbalanced, we cannot afford to continue using Lemma 4.1 (each iteration will still cost  $O(n \log s)$  to execute but  $o(s)$  edges would be found). Instead, we will treat these remaining sites differently. We say that a site is *small* if all of its edges are found while using Lemma 4.1, and *big* otherwise. By the way in which our algorithm works, small sites have  $O(n/s)$  edges in their Voronoi cell, but big ones may have many edges (if they are among the last sites to be processed they do not have necessarily many edges).

Our algorithm has three phases. In the first phase we process the whole input to detect which sites are the big ones (no edge will be reported in this phase). The second phase scans the input again and reports all edges that belong to a bisector between a small site and some other site. The third and final phase reports edges between two big sites.

**First phase.** Recall that the aim of this phase is to identify the big sites. We describe how we use Lemma 4.1 in more details. We want to scan all sites whose associated Voronoi cell is nonempty. For  $NVD(P)$ , this is trivial since all sites have a nonempty cell in  $NVD(P)$ . Hence, it suffices to scan them sequentially. The starting ray can be constructed in the same way as in Theorem 3.5. If we are interested in computing  $FVD(P)$  instead, we use the algorithm of Darwish and Elmasry [18]. This algorithm reports all sites that belong to the convex hull of  $P$  in  $O(\frac{n^2}{s \log n})$  time using  $O(s)$  words of workspace. Sites are reported one by one in clockwise order along the convex hull. Thus, we will use the output of the algorithm of Darwish and Elmasry as our input instead: we run their algorithm storing any sites that would be reported. Whenever we gathered  $s$  sites, we pause the execution of the convex hull computation and process those sites. Whenever more sites are needed, we simply resume the execution of the convex hull algorithm. Since sites are reported in clockwise order, whenever a site is reported we know both of its neighbors. This allows us to use Fact 3.2 to find a starting ray for each site.

Regardless of the order in which we process the sites, we keep  $s$  sites from  $P$  in memory. Now we apply Lemma 4.1 to compute, for every site in memory, one edge of its cell. Once the edge is computed, as in Theorem 3.5, we update the rays to look for the next edge of each cell. Whenever all edges of a cell have been found we remove the corresponding site from memory, and we insert the next site from  $P$  into the working memory. Since the Voronoi diagram has  $O(n)$  edges, and at each iteration we find  $s$  edges, after  $O(n/s)$  steps, fewer than  $s$  sites remain in memory, and all the other sites of  $P$  have been processed.

When the first phase of the algorithm ends, we identified the big sites (those remaining to be processed). Since they cannot be more than  $s$ , we store them explicitly sorted (say, in increasing index) in a table  $B$  so that membership queries can be answered in  $O(\log s)$  time.

**Second phase.** The second phase is very similar to the first one<sup>1</sup>: pick  $s$  sites to process, use Lemma 4.1 to find an edge for each site, once all edges of a site have been found, replace it with another site, and continue until only big sites remain. The main difference is that we now report every edge of the diagram that we compute provided that (i) it does not lie on a bisector between two big sites, and (ii) it has not been reported before. The second condition is detected as follows: suppose while scanning the cell of  $v_i$  we find an edge  $e$  that lies on the bisector of  $v_i$  and  $v_j$ . We report  $e$  only if either (iia) both  $v_i$  and  $v_j$  are small sites and  $i < j$ , or (iib)  $v_i$  is a small site and  $v_j$  is a big site.

**Third phase.** The aim of the third phase is to report every edge of the diagram that is on the bisector between two big sites. For this, we compute the Voronoi diagram of the big sites in  $O(s \log s)$  time. Let  $E_B$  denote the set of its edges. The edges of  $E_B$  that are also present in the Voronoi diagram of  $P$  are the ones that need to be reported (note that possibly only a portion of these edges need to be reported).

In order to confirm which edges of  $E_B$  remain in the diagram we proceed in a similar way as the second scan of Lemma 4.1: in each step we compute the Voronoi diagram of  $B$  and a batch of  $s$  sites of  $P$ . For any edge of  $E_B$ , we check whether it is cut off in the new diagram. If so, we update its endpoints in  $E_B$  and we continue with the next batch of  $s$  sites of  $P$ . After processing all the sites of  $P$ , the remaining  $O(s)$  edges in  $E_B$  that have not become empty constitute all the edges of the Voronoi diagram of  $P$  which are on a bisector of two big sites. Notice that in this procedure, in contrast to Lemma 4.1, we report  $O(s)$  edges that are not necessarily incident to  $s$  different cells.

► **Theorem 4.2.** *Let  $P = \{p_1, \dots, p_n\}$  be a planar  $n$ -point set in general position stored in a read-only array. We can report edges of  $\text{NVD}(P)$  (or  $\text{FVD}(P)$ ) in  $O((n^2/s) \log s)$  time using  $O(s)$  words of workspace.*

**Proof.** Lemma 4.1 certifies that edges reported in the second phase are part of  $\text{NVD}(P)$  (or  $\text{FVD}(P)$ ). Also, conditions (iia) and (iib) make sure that no edge is reported more than once. The reasoning for edges reported in the third phase is similar. Clearly, if an edge  $e \in \text{NVD}(P)$  (or  $e \in \text{FVD}(P)$ ) is between two big sites, the same edge (possibly a superset) must also be present in  $\text{NVD}(B)$  (or  $\text{FVD}(B)$ ). The reverse inclusion follows from exhaustiveness: for each edge of  $\text{NVD}(B)$  (or  $\text{FVD}(B)$ ) we consider all sites of  $P$  and for each one, we remove only the portion of the edge that is on the wrong side of the bisector.

Now we argue about running time. Computationally speaking, the most expensive part of the algorithm is in the  $O(n/s)$  executions of Lemma 4.1 that are done in the first and second phases. Other than that, creating table  $B$  needs  $O(s \log s)$  time, and we make  $O(n)$  lookups in  $B$ , two per edge of  $\text{NVD}(P)$  (or  $\text{FVD}(P)$ ). Each lookup needs  $O(\log s)$  time, so  $O(n \log s)$  in total. The third phase makes a single scan of the input, thus it takes  $O(n \log s)$  time. For the Farthest Voronoi algorithm we also compute the vertices of the convex hull using the approach of Darwish and Elmasry [18] for which the running time is  $o((n^2/s) \log s)$ , so non-critical.

During the execution of the algorithm we store only  $s$  sites that are currently being processed (along with  $O(1)$  information attached to each of them), the structure  $B$  of less than  $s$  big sites, the batch of  $s$  sites being processed (and its associated Voronoi diagram). All of this can be stored using  $O(s)$  words of workspace, as claimed. ◀

---

<sup>1</sup> Indeed, the first and second phases are so similar that they can be merged. However, as we will see afterwards, this is not possible for higher order Voronoi diagrams. Thus, for coherence we split the phases even for the  $k = 1$  case.

## 5 Higher-Order Voronoi Diagrams

We now consider the case of higher-order Voronoi diagrams. More precisely, we are given an integer  $K \in O(\sqrt{s})$ , and we would like to report the edges of all Voronoi diagrams of order up to  $K$ . For this, we generalize our approach from the previous section, and we combine it with a recursive procedure: for  $k = 1, \dots, K - 1$ , we compute the edges of  $\text{VD}^{k+1}(P)$  by using previously computed edges of  $\text{VD}^k(P)$ . To make efficient use of available memory, we perform the computations of  $\text{VD}^k(P)$  in a pipelined fashion, so that in each stage, the necessary edges of the previous Voronoi diagram are available.

We begin with some preliminary remarks. We call a cell  $C$  of  $\text{VD}^k(P)$  a  $k$ -cell, and we represent it as the set of  $k$  sites that are closest to all points in  $C$ . Similarly, we call a vertex  $v$  of  $\text{VD}^k(P)$  a  $k$ -vertex, and we represent it as a set of  $k + 2$  sites of  $P$ , namely  $k - 1$  sites that are closest to  $v$ , and the three sites that come next in the distance order and are equidistant to  $v$ . Finally, the edges of  $\text{VD}^k(P)$  are called  $k$ -edges. We represent them in a somewhat unusual manner: each edge of  $\text{VD}^k(P)$  is split into two directed *half-edges*, such that the half-edges are oriented in opposing directions and such that each half-edge is associated with the  $k$ -cell to its left. A half-edge  $e$  is represented by  $k + 3$  sites of  $P$ : the  $k - 1$  sites closest to  $e$ , the two sites that come next in the distance order and are equidistant to  $e$ , and two more sites needed to define the vertices at the endpoints of  $e$ . The order of the vertices encodes the direction of the half-edge. The half-edge is directed from the *tail* vertex to the *head* vertex. We will need several well-known properties of higher-order Voronoi diagrams: (I) let  $Q_1, Q_2 \subset P$  be two  $k$ -subsets such that the  $k$ -cells  $C^k(Q_1)$  and  $C^k(Q_2)$  are non-empty and adjacent (i.e., share an edge  $e$ ). Then, the set  $Q = Q_1 \cup Q_2$  has size  $k + 1$ , and  $C^{k+1}(Q)$  is a non-empty  $(k + 1)$ -cell [23]. (II) Let  $Q \subset P$  be a  $(k + 1)$ -subset such that  $C^{k+1}(Q)$  is non-empty. Then, the portion of  $\text{VD}^k(P)$  restricted to  $C^{k+1}(Q)$  is identical to (i.e., has the same vertices and edges as) the portion of  $\text{FVD}(Q)$  restricted to  $C^{k+1}(Q)$ . Furthermore, the edges of  $\text{FVD}(Q)$  in  $C^{k+1}(Q)$  do not intersect the boundary, but their endpoints either lie in the interior of  $C^{k+1}(Q)$  or coincide with vertices of  $C^{k+1}(Q)$ . Hence, every  $(k + 1)$ -cell contains at most  $O(k + 1)$   $k$ -edges and at least one  $k$ -edge, and these edges form a tree [23]. (III) Every  $k$ -vertex is either also a  $(k - 1)$ -vertex or also a  $(k + 1)$ -vertex. In particular, every vertex appears in exactly two Voronoi diagrams of consecutive order. We call a  $k$ -vertex *old*, if it is also a  $(k - 1)$ -vertex, and *new* otherwise. (All 1-vertices are new).

Next, we describe a procedure to generate all (directed)  $(k + 1)$ -half-edges, assuming that we have the  $k$ -half-edges at hand. Later, we will combine these procedures in a space-efficient manner. At a high level, our idea is as follows: let  $e$  be a  $k$ -half-edge. By property (II), the half-edge  $e$  lies inside a  $(k + 1)$ -cell  $C$ . We will see that we can use  $e$  as a starting point to report all half-edges of  $C$ , similarly as in Lemma 4.1. However, if we repeat this procedure for every  $k$ -half-edge, we may report a  $(k + 1)$ -half-edge  $\Omega(k)$  times. This will lead to problems when we combine the algorithms for computing the different orders. To avoid this, we do the following. We call a  $k$ -half-edge *relevant* if its tail vertex lies on the boundary of the  $(k + 1)$ -cell that contains it. For each  $(k + 1)$ -cell  $C$ , we partition the boundary of  $C$  into *intervals* of  $(k + 1)$ -half-edges that lie between two consecutive tail vertices of relevant  $k$ -half-edges. We assign each interval to the relevant  $k$ -half-edge of its clockwise endpoint. Now, our algorithm goes through all  $k$ -half-edges. If the current  $k$ -half-edge  $e$  is not relevant, the algorithm does nothing. Otherwise, it reports the  $(k + 1)$ -half-edges of the interval assigned to  $e$ . This ensures that every half-edge is reported exactly once. As in the previous section, we distinguish between *big* and *small* cells in  $\text{VD}^{k+1}(P)$ , lest we spend too much time on cells with many incident edges. A more detailed description follows below.

The following lemma describes an algorithm that takes  $s$   $k$ -half-edges and for each of them either determines that it is not relevant or finds the first edge of the interval of  $(k + 1)$ -half-edges assigned to it.

► **Lemma 5.1.** *Suppose we are given  $s$  different  $k$ -half-edges  $e_1^k, \dots, e_s^k$  represented by the subsets  $E_1, \dots, E_s$  of  $P$ . There is an algorithm that, for  $i = 1, \dots, s$ , either determines that  $e_i^k$  is not relevant, or finds  $e_i^{k+1}$ , the first  $(k + 1)$ -edge of the interval assigned to  $e_i^k$ . The algorithm takes total time  $O(nk \log^{1+\varepsilon} k \cdot (\log s / \log k)^{O(1)})$  and uses  $O(ks)$  words of workspace.*

**Proof.** Our algorithm proceeds analogously to Lemma 4.1. First, we inspect all  $k$ -half-edges  $e_i^k$ . If the additional site defining the tail vertex of  $e_i^k$  is one of the  $k + 1$  sites defining  $e_i^k$  (i.e., the sites which are closest in the distance order to  $e_i^k$ ), then the tail vertex of  $e_i^k$  lies in the interior of a  $(k + 1)$ -cell, and the edge is not relevant. Otherwise, the tail vertex will be an old  $(k + 1)$ -vertex, and we need to determine the first  $(k + 1)$ -half-edge for the interval assigned to  $e_i^k$ . Let  $I$  be the set of all indices  $i$  such that  $e_i^k$  is relevant.

To determine the first half-edge of each interval, we process the sites in  $P$  in batches of size  $s$ . In each iteration, we pick a new batch  $Q$  of  $s$  sites. Then, we construct  $\text{VD}^{k+1}(\bigcup_{i \in I} E_i \cup Q)$  in  $O(sk \log^{1+\varepsilon} k \cdot (\log s / \log k)^{O(1)})$  time [15]. By construction, the tail vertex of each  $e_i^k$  with  $i \in I$  belongs to the resulting diagram. Thus, we iterate over all batches, and for each  $e_i^k$ , we determine the edge  $f_i^{k+1}$  that appears in one of the resulting diagrams such that (i)  $f_i^{k+1}$  is incident to the tail vertex of  $e_i^k$ ; (ii)  $f_i^{k+1}$  is to the right of the directed line spanned by  $e_i^k$ ; and (iii) among all such edges,  $f_i^{k+1}$  makes the smallest angle with  $e_i^k$ . We need  $O(n/s)$  iterations to find  $f_i^{k+1}$ . Now, for each  $i \in I$ , the desired  $(k + 1)$ -half-edge  $e_i^{k+1}$  is a subset of  $f_i^{k+1}$ . Thus, as in Lemma 4.1, we perform a second scan over  $P$  to find the other endpoint of  $e_i^{k+1}$ . We orient  $e_i^{k+1}$  such that the cell containing  $e_i^k$  lies to the left of it.

It follows that we can process  $s$  edges of  $\text{VD}^k(P)$  in  $O(nk \log^{1+\varepsilon} k \cdot (\log s / \log k)^{O(1)})$  time using a workspace with  $O(ks)$  words to store the  $s$  different subsets for the edges. ◀

The algorithm from Lemma 5.1 is actually more general. If, instead of a  $k$ -half-edge  $e_i^k$  that lies inside a  $(k + 1)$ -cell  $C$ , we have a  $(k + 1)$ -half-edge  $e_i^{k+1}$  that lies on the boundary of  $C$ , the same method of processing  $P$  in batches of size  $s$  allows us to find the next  $(k + 1)$ -half-edge incident to  $C$  in counterclockwise order from  $e_i^{k+1}$ . These two kinds of edges can be handled simultaneously.

► **Corollary 5.2.** *Suppose we are given  $s$  half-edges  $e_1, \dots, e_s$  such that each  $e_i$  is either a  $k$ -half-edge or a  $(k + 1)$ -half-edge. Then, we can find in total time  $O(nk \log^{1+\varepsilon} k \cdot (\log s / \log k)^{O(1)})$  and using  $O(ks)$  words of workspace a sequence  $f_1, \dots, f_s$  of  $(k + 1)$ -half-edges such that, for  $i = 1, \dots, s$ , we have*

1. if  $e_i$  is a relevant  $k$ -half-edge, then  $f_i$  is the first  $(k + 1)$ -half-edge of the interval for  $e_i$ ;
2. if  $e_i$  is  $k$ -half-edge-that is not relevant, then  $f_i = \perp$ ; or
3. if  $e_i$  is a  $(k + 1)$ -half-edge, then  $f_i$  is the counterclockwise successor of  $e_i$ .

► **Lemma 5.3.** *Using two scans over all  $k$ -half-edges, we can report the  $(k + 1)$ -half-edges in batches of size at most  $s$  such that each  $(k + 1)$ -half-edge is reported exactly once. This takes  $O(\frac{n^2 k^2}{s} \log^{1+\varepsilon} k \cdot (\log s / \log k)^{O(1)})$  time using  $O(ks)$  words of workspace.*

**Proof.** The algorithm consists of three phases: In the first phase, we keep  $s$  half-edges  $e_1, \dots, e_s$  such that each  $e_i$  is either a  $k$ -half-edge or a  $(k + 1)$ -half-edge. In each iteration, we apply Corollary 5.2 to these half-edges, to obtain  $s$  new  $(k + 1)$ -half-edges  $f_1, \dots, f_s$ . Now, for each  $i = 1, \dots, s$ , three cases apply: (i)  $f_i = \perp$ , i.e.,  $e_i$  was not relevant. In the next iteration,

we replace  $e_i$  with a fresh  $k$ -half-edge; (ii)  $f_i \neq \perp$ . Now we need to determine whether  $f_i$  is the last half-edge of its interval. For this, we check whether the head vertex of  $f_i$  is a new  $(k+1)$ -vertex. As in Lemma 5.1, this can be done by checking whether the additional site that determines the head vertex of  $f_i$  is among the  $k+2$  sites determining  $f_i$ . If  $f_i$  is not the last half-edge of its interval, we set  $e_i$  to  $f_i$  for the next iteration; otherwise, we set  $e_i$  to a fresh  $k$ -half-edge. We repeat this procedure until there are no fresh  $k$ -half-edges left.

The remaining  $k$ -half-edges in the working memory are incident to the big  $(k+1)$ -cells. We store them in an array  $B^{k+1}$ , sorted according to the lexicographic order of the indices of their sites. We emphasize that in the first phase, we do not report any  $(k+1)$ -edge.

In the second phase, we repeat the same procedure as in the first phase. However, this time we report (i) every  $(k+1)$ -half-edge incident to a small  $(k+1)$ -cell; and (ii) the opposite direction of every  $(k+1)$ -half-edge  $e$  incident to a small  $(k+1)$ -cell, so that the  $(k+1)$ -cell on the right of  $e$  is a big  $(k+1)$ -cell. We use  $B^{k+1}$  to identify the big cells.

In the third phase we report every  $(k+1)$ -half-edge  $e$  that is incident to a big  $(k+1)$ -cell, while the  $(k+1)$ -cell on the right of  $e$  is also a big  $(k+1)$ -cell. Let  $\{B^{k+1}\}$  denote the sites that define the big  $(k+1)$ -cells. We construct  $\text{VD}^{k+1}(\{B^{k+1}\})$  in the working memory. Then, we go through the sites in  $P$  in batches of size  $s$ , adding the sites of each batch to  $\text{VD}^{k+1}(\{B^{k+1}\})$ . While doing this, as in the algorithm for Lemma 4.2, we keep track of how the edges of  $\text{VD}^{k+1}(\{B^{k+1}\})$  are cut by the new diagrams. In the end, we report all  $(k+1)$ -edges of  $\text{VD}^{k+1}(\{B^{k+1}\})$  that are not empty. By *report*, we mean report two  $(k+1)$ -half-edges in opposing directions. As we explained in the algorithm for Lemma 4.2, these  $(k+1)$ -half-edges cover all the  $(k+1)$ -half-edges incident to a big  $(k+1)$ -cell, while their right cell is also a big  $(k+1)$ -cell.

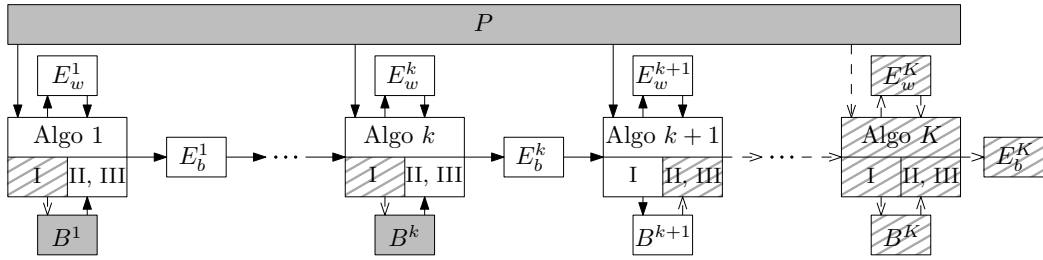
Regarding the running time, the first and the second phase consists of  $O(nk/s)$  applications of Corollary 5.2 which takes  $O(\frac{n^2k^2}{s} \log^{1+\varepsilon} k \cdot (\log s / \log k)^{O(1)})$  time. Sorting the big  $(k+1)$ -cells in  $B^{k+1}$  takes  $O(ks(\log k + \log s))$  steps: we sort the indices of the sites of each big  $(k+1)$ -cell in  $O(k \log k)$  steps. Then we sort the big cells, where each comparison in the lexicographic order requires  $O(k)$  steps, for a total of  $O(ks \log s)$  steps.

A query in  $B^{k+1}$  takes  $O(k \log k + \log s)$  time: given a query  $(k+1)$ -cell  $C$  we sort its indices in  $O(k \log k)$  time. Then we use binary-search to find cells in  $B^{k+1}$  with the same first index as  $C$ . Among these, we continue the binary-search with comparing the second indices, and so on. Thus, in each step we compare only one index of two  $(k+1)$ -cells, and either the size of search set is halved, or the search continues with the next index of  $C$ . Thus, searching a cell  $C$  with sorted index in  $B^{k+1}$  requires  $O(k + \log s)$  time.

The algorithm performs at most two searches in  $B^{k+1}$  per each  $(k+1)$ -half-edge, for a total of  $O(nk)$  edges. In the third phase, constructing a  $(k+1)$ -order Voronoi diagram of  $O(ks)$  sites takes  $O(sk \log^{1+\varepsilon} k \cdot (\log s / \log k)^{O(1)})$  time. We repeat it  $O(n/s)$  times, which takes  $O(nk \log^{1+\varepsilon} k \cdot (\log s / \log k)^{O(1)})$  time in total.

Overall, the running time of the algorithm simplifies to  $O(\frac{n^2k^2}{s} \log^{1+\varepsilon} k \cdot (\log s / \log k)^{O(1)})$ . The algorithm uses a workspace of  $O(sk)$  words, for running Corollary 5.2, for storing big  $(k+1)$ -cells and for constructing Voronoi diagrams of size  $O(ks)$ . ◀

Now, in order to find  $k$ -half-edges for all  $k = 1, \dots, K$ , we proceed as follows: first, we compute  $s$  1-edges (notice that we report every 1-edge as two 1-half-edges in opposing directions). Then, we apply Lemma 5.3 in a pipelined fashion to obtain  $k$ -half-edges for all  $k = 2, \dots, K$ . In each iteration, the algorithm from Lemma 5.3 consumes at most  $s$   $k$ -half-edges from the previous order and produces at most  $2s$   $(k+1)$ -half-edges to be used at the next order. This means that if we have between  $s$  and  $3s$  new  $k$ -half-edges available in a buffer, then we can use them one by one whenever the algorithm for computing  $k$ -half-edges



■ **Figure 2** This diagram shows the algorithm in *stage k*. For  $i = 1, \dots, K$ , Algo  $i$  is the algorithm for computing  $i$ -edges. The roman numerals I, II and III refer, respectively, to the first, second and third phase of Algo  $i$ . The white and the tiled rectangles are, respectively, active and inactive parts in stage  $k$  of the main algorithm.  $E_w^i$ ,  $E_b^i$  and  $B^i$  indicate memory cells, and they are, respectively, the working memory of Algo  $i$ , the buffer for  $i$ -edges, and the big  $i$ -cells. The algorithm in stage  $k$ , does not use the tiled memory cells, and it uses the gray ones only for reading the data that has been produced in previous stages. The arrows show reading from or writing to memory cells, and the dashed arrows are inactive in stage  $k$ . In stage  $k$ , all the  $k$ -half-edges are reported and the big  $(k + 1)$ -cells are inserted into  $B^{k+1}$  for being used in the next stages.

in Lemma 5.3 requires such a new  $k$ -half-edge. Whenever a buffer falls below  $s$  half-edges, we run the algorithm for the previous order until the buffer size is again between  $s$  and  $3s$ . Applying this idea for all the orders  $k = 1, \dots, K - 1$ , we need to store  $K - 1$  buffers, each containing up to  $3s$  half-edges for the corresponding diagram. Furthermore, for each diagram, we need to store the current workspace required by the algorithm to produce new edges (as in Lemma 5.3). Since a  $k$ -edge is represented by  $O(k)$  sites from  $P$ , the buffer for  $k$ -edges requires  $O(ks)$  words of memory. We denote it by  $E_b^k$ . Furthermore, by Lemma 5.3, the new  $k$ -edges can be found using  $O(ks)$  words of workspace, which we denote by  $E_w^k$ .

► **Theorem 5.4.** *Let  $K \in O(\sqrt{s})$  and  $P = \{p_1, \dots, p_n\}$  be a planar  $n$ -point set in general position, given in a read-only array. We can report all the edges of  $\text{VD}^1(P), \dots, \text{VD}^K(P)$  in  $O(\frac{n^2 K^6}{s} \log^{1+\varepsilon} K \cdot (\log s / \log K)^{O(1)})$  time using a workspace of size  $O(s)$ .*

**Proof.** We compute the half-edges of  $\text{VD}^1(P), \dots, \text{VD}^K(P)$  simultaneously, in a pipelined fashion. For  $k = 1$  we use the algorithm of Theorem 4.2 and for  $k = 2, \dots, K$ , we run the algorithm from Lemma 5.3 to compute  $k$ -edges. The algorithm for computing  $\text{VD}^k(P)$ , has its own working memory, denoted by  $E_w^k$  and an output buffer  $E_b^k$ . In addition, it has an array  $B^k$  to store the big  $k$ -cells for being used in the second and third phase of the algorithm. Each of these arrays should be able to store  $O(s')$  half-edges and cells of  $\text{VD}^k(P)$ , for  $s' = s/K^2$ . Since we need  $O(k)$  sites to represent a  $k$ -half-edge or a  $k$ -cell, the total space requirement for all algorithms is  $O(s)$ .

We now describe how the simultaneous algorithms interact. Our algorithm works in *stages*. In stage 0, we perform only the first phase of Theorem 4.2, to find the  $O(s')$  big cells of  $\text{VD}^1(P)$ , and we store them in  $B^1$ . Now we know the big 1-cells. Then, in stage 1, we perform the second phase of Theorem 4.2 to find and report the half-edges of  $\text{VD}^1(P)$  in batches of size at most  $2s'$ , and we store these 1-half-edges in  $E_b^1$ . Whenever we have at least  $s'$  half-edges in  $E_b^1$ , we pause the algorithm of Theorem 4.2, and we perform the first phase of Lemma 5.3 to find half-edges of  $\text{VD}^2(P)$  with  $E_b^1$  as input. Whenever the algorithm for  $\text{VD}^2(P)$  requires new 1-half-edges, and the buffer  $E_b^1$  falls below  $s'$  half-edges, we continue running the algorithm for  $\text{VD}^1(P)$ . When the algorithm for  $\text{VD}^2(P)$  has consumed all 1-half-edges and there are less than  $s'$  half-edges in  $E_w^2$ , then we stop the algorithm for  $\text{VD}^2(P)$ . The half-edges in  $E_w^2$  represent the big cells of  $\text{VD}^2(P)$ , and we store them in  $B^2$ . This concludes stage 1.



In general, in stage  $k$  of the algorithm, we identified the big cells  $B^1, \dots, B^k$  of the first  $k$  diagrams. We perform the second and the third phase of Theorem 4.2 and Lemma 5.3 in a pipelined fashion to generate the half-edges of  $VD^1(P), \dots, VD^k(P)$  and store them in the buffers  $E_b^1, \dots, E_b^k$ . We report the half-edges of  $VD^k(P)$  and if  $k \neq K$ , we also use  $E_b^k$  as an input of the first phase of Lemma 5.3, which gives us  $B^{k+1}$  for the next stage; see Figure 2. Using this procedure, we report every half-edge of  $VD^1(P), \dots, VD^K(P)$  exactly once.

Regarding the running time, in each stage  $k = 1, \dots, K$ , we have to compute all diagrams  $VD^1(P), \dots, VD^k(P)$ , using Lemma 5.3. This takes  $O\left(\frac{n^2 k^3}{s'} \log^{1+\varepsilon} k \cdot (\log s' / \log k)^{O(1)}\right)$  time. The running time for stage 0 is negligible. The complete algorithm takes  $O\left(\frac{n^2 K^4}{s'} \log^{1+\varepsilon} K \cdot (\log s' / \log K)^{O(1)}\right)$  time, which is  $O\left(\frac{n^2 K^6}{s} \log^{1+\varepsilon} K \cdot (\log s / \log K)^{O(1)}\right)$ , in terms of  $s$ . ◀

**Acknowledgements.** The authors would like to thank Luis Barba, Kolja Junginger, Elena Khramtcova, and Evanthia Papadopoulou for fruitful discussions on this topic.

---

## References

- 1 Alok Aggarwal, Leonidas J. Guibas, James B. Saxe, and Peter W. Shor. A linear-time algorithm for computing the voronoi diagram of a convex polygon. *Discrete Comput. Geom.*, 4:591–604, 1989.
- 2 Boris Aronov, Matias Korman, Simon Pratt, André van Renssen, and Marcel Roeloffzen. Time-space trade-offs for triangulating a simple polygon. In *Proc. 15th Scand. Sympos. Workshops Algorithm Theory (SWAT)*, volume 53, pages 30:1–30:12, 2016.
- 3 Sanjeev Arora and Boaz Barak. *Computational Complexity. A modern approach*. Cambridge University Press, 2009.
- 4 Tetsuo Asano, Kevin Buchin, Maike Buchin, Matias Korman, Wolfgang Mulzer, Günter Rote, and André Schulz. Memory-constrained algorithms for simple polygons. *Comput. Geom.*, 46(8):959–969, 2013.
- 5 Tetsuo Asano and David Kirkpatrick. Time-space tradeoffs for all-nearest-larger-neighbors problems. In *Proc. 13th Int. Symp. Algorithms and Data Structures (WADS)*, pages 61–72, 2013.
- 6 Tetsuo Asano, Wolfgang Mulzer, Günter Rote, and Yajun Wang. Constant-work-space algorithms for geometric problems. *J. of Comput. Geom.*, 2(1):46–68, 2011.
- 7 Tetsuo Asano, Wolfgang Mulzer, and Yajun Wang. Constant-work-space algorithms for shortest paths in trees and simple polygons. *J. Graph Algorithms Appl.*, 15(5):569–586, 2011.
- 8 Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi diagrams and Delaunay triangulations*. World Scientific Publishing Co. Pte. Ltd., Hackensack, NJ, 2013.
- 9 Yeganeh Bahoo, Bahareh Banyassady, Prosenjit Bose, Stephane Durocher, and Wolfgang Mulzer. Finding the  $k$ -visibility region of a point in a simple polygon in the memory-constrained model. In *Proc. 32nd European Workshop Comput. Geom. (EWCG)*, 2016.
- 10 Luis Barba, Matias Korman, Stefan Langerman, Kunihiko Sadakane, and Rodrigo I. Silveira. Space–time trade-offs for stack-based algorithms. *Algorithmica*, 72(4):1097–1129, 2015.
- 11 Luis Barba, Matias Korman, Stefan Langerman, and Rodrigo I. Silveira. Computing the visibility polygon using few variables. *Comput. Geom.*, 47(9):918–926, 2013.
- 12 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational geometry. Algorithms and applications*. Springer-Verlag, third edition, 2008.
- 13 Allan Borodin and Stephen A. Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM J. Comput.*, 11:287–297, 1982.

- 14 Hervé Brönnimann, Timothy M. Chan, and Eric Y. Chen. Towards in-place geometric algorithms and data structures. In *Proc. 20th Annu. Sympos. Comput. Geom. (SoCG)*, pages 239–246, 2004.
- 15 Timothy M Chan. Remarks on  $k$ -level algorithms in the plane. *manuscript, Univ. of Waterloo*, 1999.
- 16 Timothy M Chan. Random sampling, halfspace range reporting, and construction of  $(\leq k)$ -levels in three dimensions. *SIAM J. Comput.*, 30(2):561–575, 2000.
- 17 Timothy M. Chan and Eric Y. Chen. Multi-pass geometric algorithms. *Discrete Comput. Geom.*, 37(1):79–102, 2007.
- 18 Omar Darwish and Amr Elmasry. Optimal time-space tradeoff for the 2D convex-hull problem. In *Proc. 22nd Annu. European Sympos. Algorithms (ESA)*, pages 284–295, 2014.
- 19 Amr Elmasry and Frank Kammer. Space-efficient plane-sweep algorithms. *arXiv : 1507.01767*, 2015.
- 20 Sarel Har-Peled. Shortest path in a polygon using sublinear space. *J. of Comput. Geom.*, 7(2):19–45, 2016.
- 21 Matias Korman. Memory-constrained algorithms. In *Encyclopedia of Algorithms*, pages 1260–1264. Springer Berlin Heidelberg, 2016. doi:10.1007/978-3-642-27848-8\_586-1.
- 22 Matias Korman, Wolfgang Mulzer, André van Renssen, Marcel Roeloffzen, Paul Seiferth, and Yannik Stein. Time-space trade-offs for triangulations and Voronoi diagrams. In *Proc. 14th Int. Symp. Algorithms and Data Structures (WADS)*, pages 482–494, 2015.
- 23 Der-Tsai Lee. On  $k$ -nearest neighbor Voronoi diagrams in the plane. *IEEE Trans. Computers*, 31(6):478–487, 1982.
- 24 J. Ian Munro and Mike Paterson. Selection and sorting with limited storage. *Theoret. Comput. Sci.*, 12:315–323, 1980.
- 25 J. Ian Munro and Venkatesh Raman. Selection from read-only memory and sorting with minimum data movement. *Theoret. Comput. Sci.*, 165(2):311–323, 1996.
- 26 J. Pagter and T. Rauhe. Optimal time-space trade-offs for sorting. In *Proc. 39th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 264–268, 1998.
- 27 Ira Pohl. A minimum storage algorithm for computing the median. Technical Report RC2701, IBM, 1969.
- 28 John E. Savage. *Models of computation – exploring the power of computing*. Addison-Wesley, 1998.

# Energy-Efficient Delivery by Heterogeneous Mobile Agents<sup>\*†</sup>

Andreas Bärtschi<sup>1</sup>, Jérémie Chalopin<sup>2</sup>, Shantanu Das<sup>3</sup>,  
Yann Disser<sup>4</sup>, Daniel Graf<sup>5</sup>, Jan Hackfeld<sup>6</sup>, and Paolo Penna<sup>7</sup>

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

2 LIF, CNRS and Aix-Marseille Université, Marseille, France  
jeremie.chalopin@lif.univ-mrs.fr

3 LIF, CNRS and Aix-Marseille Université, Marseille, France  
shantanu.das@lif.univ-mrs.fr

4 TU Darmstadt, Germany  
disser@mathematik.tu-darmstadt.de

5 Department of Computer Science, ETH Zürich, Zürich, Switzerland  
daniel.graf@inf.ethz.ch

6 Institut für Mathematik, TU Berlin, Berlin, Germany  
hackfeld@math.tu-berlin.de

7 Department of Computer Science, ETH Zürich, Zürich, Switzerland  
paolo.penna@inf.ethz.ch

---

## Abstract

We consider the problem of delivering  $m$  messages between specified source-target pairs in an undirected graph, by  $k$  mobile agents initially located at distinct nodes of the graph. Each edge has a designated length and each agent consumes energy proportional to the distance it travels in the graph. We are interested in optimizing the total energy consumption for the team of agents. Unlike previous related work, we consider heterogeneous agents with different rates of energy consumption (weights  $w_i$ ). To solve the delivery problem, agents face three major challenges: *Collaboration* (how to work together on each message), *Planning* (which route to take) and *Coordination* (how to assign agents to messages).

We first show that the delivery problem can be 2-approximated *without* collaborating and that this is best possible, i.e., we show that the *benefit of collaboration* is 2 in general. We also show that the benefit of collaboration for a single message is  $1/\ln 2 \approx 1.44$ . Planning turns out to be NP-hard to approximate even for a single agent, but can be 2-approximated in polynomial time if agents have unit capacities and do not collaborate. We further show that coordination is NP-hard even for agents with unit capacity, but can be efficiently solved exactly if they additionally have uniform weights. Finally, we give a polynomial-time  $(4 \max \frac{w_i}{w_j})$ -approximation for message delivery with unit capacities.

**1998 ACM Subject Classification** F.2 Analysis of Algorithms and Problem Complexity, F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** message delivery, mobile agents, energy optimization, approximation algorithms

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.10

---

\* A full version of the paper is available at <https://arxiv.org/abs/1610.02361>

† This work was partially supported by the project ANR-ANCOR (anr-14-CE36-0002-01), the SNF (project 200021L\_156620) and the DFG Priority Programme 1736 “Algorithms for Big Data” (grant SK 58/10-1).



© Andreas Bärtschi, Jérémie Chalopin, Shantanu Das, Yann Disser, Daniel Graf, Jan Hackfeld, and Paolo Penna;  
licensed under Creative Commons License CC-BY

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 10; pp. 10:1–10:14

Leibniz International Proceedings in Informatics



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

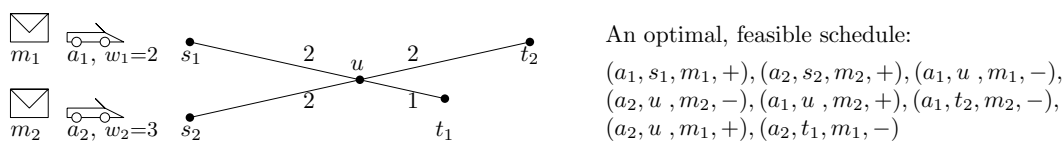
## 1 Introduction

Recent technological progress in robotics allows the mass production of inexpensive mobile robots which can be used to perform a variety of tasks autonomously without the need for human intervention. This gives rise to a variety of algorithmic problems for teams of autonomous robots, hereafter called *mobile agents*. We consider here the delivery problem of moving some objects or messages between various locations. A mobile agent corresponds to an automated vehicle that can pick up a message at its source and deliver it to the intended destination. In doing so, the agent consumes energy proportional to the distance it travels. Our goal is to design a centralized algorithm for the agents such that the total energy consumed is minimized.

In general the agents may not be all identical; some may be more energy efficient than others if they use different technologies or different sources of power. We assume each agent has a given *weight* which is the rate of energy consumption per unit distance traveled by this agent (here we use the term *weights*, since the rates are weights of the objective function). Moreover, the agents may start from distinct locations. Thus it may be sometimes efficient for an agent to carry the message to some intermediate location and hand it over to another agent which carries it further towards the destination. On the other hand, an agent may carry several messages at the same time. Finding an optimal solution that minimizes the total energy cost involves scheduling the moves of the agents and the points where they pick up or handover the messages. We study this problem (called WEIGHTEDDELIVERY) for a graph  $G$  which connects all sources and destinations. The objective is to deliver  $m$  messages between specific source-target pairs using  $k$  agents located in arbitrary nodes of  $G$ . Note that this problem is distinct from the connectivity problems on graphs or network flow problems since the initial location of the agents are in general different from the sources where the messages are located, which means we need to consider the cost of moving the agents to the sources in addition to the cost of moving the messages. Furthermore, there is no one-to-one correspondence between the agents and the messages in our problem.

Previous approaches to energy-efficient delivery of messages by agents have focused on a bottleneck where the agents have limited energy (battery power) which restricts their movements [1, 8]. The decision problem of whether a single message can be delivered without exceeding the available energy for any agent is known as the DataDelivery problem [9] or the BudgetedDelivery problem [4] and it was shown to be weakly NP-hard on paths [9] and strongly NP-hard on planar graphs [4].

**Our Model.** We consider an undirected graph  $G = (V, E)$ . Each edge  $e \in E$  has a *cost* (or *length*) denoted by  $l_e$ . The length of a simple path is the sum of the lengths of its edges. The distance between nodes  $u$  and  $v$  is denoted by  $d_G(u, v)$  and is equal to the length of the shortest path from  $u$  to  $v$  in  $G$ . There are  $k$  mobile agents denoted by  $a_1, \dots, a_k$  and having weights  $w_1, \dots, w_k$ . These agents are initially located on arbitrary nodes  $p_1, \dots, p_k$  of  $G$ . We denote by  $d(a_i, v)$  the distance from the initial location of  $a_i$  to node  $v$ . Each agent can move along the edges of the graph. Each time an agent  $a_i$  traverses an edge  $e$  it incurs an energy cost of  $w_i \cdot l_e$ . Furthermore there are  $m$  pairs of (source, target) nodes in  $G$  such that for  $1 \leq i \leq m$ , a message has to be delivered from source node  $s_i$  to a target node  $t_i$ . A message can be picked up by an agent from any node that it visits and it can be carried to any other node of  $G$ , and dropped there. The agents are given a *capacity*  $\kappa$  which limits the number of messages an agent may carry simultaneously. There are no restrictions on how much an agent may travel. We denote by  $d_j$  the total distance traveled by the  $j$ -th



■ **Figure 1** Example of an optimal, feasible schedule for two messages and two agents.

agent. **WEIGHTEDDELIVERY** is the optimization problem of minimizing the total energy  $\sum_{j=1}^k w_j \cdot d_j$  needed to deliver all messages.

A *schedule*  $S$  describes the actions of all agents as a sequence (ordered list) of pick-up actions  $(a_j, p, m_i, +)$  and drop-off actions  $(a_j, q, m_i, -)$ , where each such tuple denotes the action of agent  $a_j$  moving from its current location to node  $p$  (node  $q$ ) where it picks up message  $m_i$  (drops message  $m_i$ , respectively). A schedule  $S$  implicitly encodes all the pick-up and drop-off times and it is easy to compute its total energy use of  $\text{COST}(S) := \sum_{j=1}^k w_j d_j$ . We denote by  $S|_{a_j}$  the subsequence of all actions carried out by agent  $a_j$  and by  $S|_{m_i}$  the subsequence of all actions involving pick-ups or drop-offs of message  $m_i$ . We call a schedule *feasible* if every pick-up action  $(\_, p, m_i, +)$ ,  $p \neq s_i$ , is directly preceded by a drop-off action  $(\_, p, m_i, -)$  in  $S|_{m_i}$  and if all the messages get delivered, see Figure 1.

**Our Contribution.** Solving **WEIGHTEDDELIVERY** naturally involves simultaneously solving three subtasks, *collaboration*, *individual planning*, and *coordination*: First of all, if multiple agents work on the same message, they need to collaborate, i.e., we have to find all intermediate drop-off and pick-up locations of the message. Secondly, if an agent works on more than one message, we have to plan in which order it wants to approach its subset of messages. Finally, we have to coordinate which agent works on which subset of all messages (if they do this without collaboration, the subsets form a partition, otherwise the subsets are not necessarily pairwise disjoint). Even though these three subtasks are interleaved, we investigate collaboration, planning and coordination separately in the next three sections. This leads us to a polynomial-time approximation algorithm for **WEIGHTEDDELIVERY**, given in Section 5.

In Section 2 we consider the *Collaboration* aspect of **WEIGHTEDDELIVERY**. We first present a polynomial time solution for **WEIGHTEDDELIVERY** when there is only a single message ( $m = 1$ ). The algorithm has complexity  $O(|V|^3)$  irrespective of the number of agents  $k$ . In general, we show that any algorithm that only uses one agent for delivering every message cannot achieve an approximation ratio better than what we call the *benefit of collaboration* (BoC) which is at least  $1/\ln((1 + 1/(2m))^m (1 + 1/(2m + 1)))$ . We show this to be tight for  $m = 1$  (where  $\text{BoC} \geq 1/\ln 2$ ) and  $m \rightarrow \infty$  (where  $\text{BoC} \rightarrow 2$ ).

In Section 3 we look at the *Planning* aspect of **WEIGHTEDDELIVERY**. Individual planning by itself turns out to be NP-hard on planar graphs and NP-hard to approximate within a factor of less than  $\frac{367}{366}$ . On the positive side, we give approximation guarantees for restricted versions of **WEIGHTEDDELIVERY** which turn out to be useful for the analysis in Section 5.

In Section 4 we study the *Coordination* aspect of **WEIGHTEDDELIVERY**. Even if collaboration and planning are taken care of (i.e., a schedule is fixed except for the assignment of agents to messages), Coordination also turns out to be NP-hard even on planar graphs. The result holds for any capacity, including  $\kappa = 1$ . This setting, however, becomes tractable if restricted to uniform weights of the agents.

In Section 5 we give a polynomial-time approximation algorithm for **WEIGHTEDDELIVERY** with an approximation ratio of  $4 \cdot \max \frac{w_i}{w_j}$  for  $\kappa = 1$ . Due to the limited space, some proofs are omitted, but can be found in the full version of the paper [6].

**Related Work.** The problem of communicating or transporting goods between sources and destinations in a graph has been well studied in a variety of models with different optimization criteria. The problem of finding the smallest subgraph or tree that connects multiple sources and targets in a graph is called the *point-to-point connection problem* and is known to be NP-hard [25]. The problem is related to the more well-known generalized Steiner tree problem [28] which is also NP-hard. Unlike these problems, the maximum flow problem in a network [14], puts a limit on the number of messages that can be transported over an edge, which makes the problem easier allowing for polynomial time solutions. In all these problems, however, there are no agents carrying the messages as in our problem.

For the case of a single agent moving in a graph, the task of optimally visiting all nodes, called the *Traveling salesman problem* or visiting all edges, called the *Chinese postman problem* have been studied before. The former is known to be NP-hard [2] while the latter can be solved in  $O(|V|^2|E|)$  time [13]. For metric graphs, the traveling salesman problem has a polynomial-time  $\frac{3}{2}$ -approximation for tours [10] and for paths with one fixed endpoint [18]. For multiple identical agents in a graph, Demaine et al. [12] studied the problem of moving the agents to form desired configurations (e.g. connected or independent configurations) and they provided approximation algorithms and inapproximability results. Bilo et al. [7] studied similar problems on visibility graphs of simple polygons and showed many motion planning problems to be hard to approximate.

Another optimization criteria is to minimize the maximum energy consumption by any agent, which requires partitioning the given task among the agents. Frederickson et al. [17] studied this for uniform weights and called it the *k-stacker-crane problem* and they gave approximation algorithms for a single agent and multiple agents. Also in this minmax context, the problem of visiting all the nodes of a tree using  $k$  agents starting from a single location is known to be NP-hard [16]. Anaya et al. [1] studied the model of agents having limited energy budgets. They presented hardness results (on trees) and approximation algorithms (on arbitrary graphs) for the problem of transferring information from one agent to all others (*Broadcast*) and from all agents to one agent (*Convergecast*). For the same model, message delivery between a single  $s$ - $t$  node pair was studied by Chalopin et al. [8, 9, 4] as mentioned above. A recent paper [11] shows that these three problems remain NP-hard for general graphs even if the agents are allowed to exchange energy when they meet.

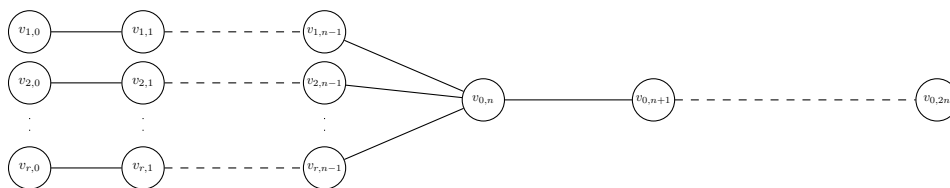
## 2 Collaboration

In this section, we examine the *collaboration* of agents: Given for each message  $m_i$  all the agents  $a_{i1}, a_{i2}, \dots, a_{ix}$  which at some point carry the message, we need to find all pick-up and drop-off locations (handovers)  $h_1, \dots, h_y$  for the schedule entries  $(a_{i1}, \_, m_i, +)$ ,  $(a_{i1}, \_, m_i, -), \dots, (a_{ix}, \_, m_i, -)$ . Note, that in general we can have more than two action quadruples  $(a_{ij}, \_, m_i, +/-)$  per agent  $a_{ij}$ . When there is only a single message overall ( $m = 1$ ), we will use a structural result to tie together WEIGHTEDDELIVERY and Collaboration. For multiple messages, however, this no longer holds: In this case, we analyze the benefit we lose if we forego collaboration and deliver each message with a single agent.

### 2.1 An Algorithm for WeightedDelivery of a Single Message

► **Lemma 1.** *In any optimal solution to WEIGHTEDDELIVERY for a single message, if the message is delivered by agents with weights  $w_1, w_2, \dots, w_k$ , in this order, then*

- (i)  $w_i \geq w_j$  whenever  $i < j$ , and
- (ii) without loss of generality,  $w_i \neq w_j$  for  $i \neq j$ .



■ **Figure 2** Lower bound construction for the benefit of collaboration.

Hence there is an optimal schedule  $S$  in which no agent  $a_j$  has more than one pair of pick-up/drop-off actions.

► **Theorem 2.** *An optimal solution of WEIGHTEDDELIVERY of a single message in a graph  $G = (V, E)$  with  $k \leq |V|$  agents can be found in  $O(|V|^3)$  time.*

**Proof.** We use the properties of Lemma 1 to create an auxiliary graph on which we run Dijkstra’s algorithm for computing a shortest path from  $s$  to  $t$ . Given an original instance of single-message WEIGHTEDDELIVERY consisting of the graph  $G = (V, E)$ , with  $s, t \in V$ , we obtain the auxiliary, *directed* graph  $G' = (V', E')$  as follows:

- For each node  $v \in V$  and each agent  $a_i$ , there is a node  $v_{a_i}$  in  $G'$ . Furthermore  $G'$  contains two additional vertices  $s$  and  $t$ .
  - For  $1 \leq i \leq k$ , there is an arc  $(s, s_{a_i})$  of cost  $w_i \cdot d_G(p_i, s)$  and an arc  $(t_{a_i}, t)$  of cost 0.
  - For  $(u, v) \in E$  and  $1 \leq i \leq k$ , there are two arcs  $(u_{a_i}, v_{a_i})$  and  $(v_{a_i}, u_{a_i})$  of cost  $w_i \cdot l_{(u,v)}$ .
  - For  $u \in V$  and agents  $a_i, a_j$  with  $w_i > w_j$ , there is an arc  $(u_{a_i}, u_{a_j})$  of cost  $w_j \cdot d_G(p_j, u)$ .
- Note that any solution to the WEIGHTEDDELIVERY that satisfies the properties of Lemma 1 corresponds to some  $s$ - $t$ -path in  $G'$  such that the cost of the solution is equal to the length of this path in  $G'$  and vice versa. This implies that the length of the shortest  $s$ - $t$  path in  $G'$  is the cost of the optimal solution for WEIGHTEDDELIVERY in  $G$ . Assuming that  $k \leq |V|$ , the graph  $G'$  has  $|V| \cdot k + 2 \in O(|V|^2)$  vertices and at most  $2k + (k^2|V| + |V|^2k)/2 - |V| \cdot k \in O(|V|^3)$  arcs. The edge-costs of the graph  $G'$  can be computed in  $O(|V|^3)$  time if we use the *Floyd Warshall* all pair shortest paths algorithm [15, 27] in  $G$ . Finally, we compute the shortest path from  $s$  to  $t$  in  $G'$  in time  $O(|V|^3)$ , using Dijkstra’s algorithm with Fibonacci heaps. ◀

Unfortunately, the structural properties of Lemma 1 do not extend to multiple messages. In the next two subsections we investigate how the quality of an optimal solution changes if we only allow every message to be transported by one agent. Different messages may still be transported by different agents and one agent may also transport multiple messages at the same time as long as the number of messages is at most the capacity  $\kappa$ . To this end we define the *Benefit of Collaboration* as the cost ratio between an optimal schedule OPT and a best-possible schedule without collaboration  $S$ ,  $\text{BOC} = \min_S \text{COST}(S)/\text{COST}(\text{OPT})$ .

## 2.2 Lower Bound on the Benefit of Collaboration

► **Theorem 3.** *On instances of WEIGHTEDDELIVERY with agent capacity  $\kappa$  and  $m$  messages, an algorithm using one agent for delivering every message cannot achieve an approximation ratio better than  $1/\ln((1 + 1/(2r))^r (1 + 1/(2r + 1)))$ , where  $r := \min\{\kappa, m\}$ .*

**Proof.** Consider the graph  $G = (V, E)$  given in Figure 2, where the length  $l_e$  of every edge  $e$  is  $1/n$ . This means that  $G$  is a star graph with center  $v_{0,n}$  and  $r + 1$  paths of total length 1 each. We have  $r$  messages and message  $i$  needs to be transported from  $v_{i,0}$  to  $v_{0,2n}$  for

## 10:6 Energy-Efficient Delivery by Heterogeneous Mobile Agents

$i = 1, \dots, r$ . There further is an agent  $a_{i,j}$  with weight  $w_{i,j} = \frac{2r}{2r+j/n}$  starting at every vertex  $v_{i,j}$  for  $(i, j) \in \{1, \dots, r\} \times \{0, \dots, n-1\} \cup \{0\} \times \{n, \dots, 2n\}$ .

We first show the following: If any agent transports  $s$  messages  $i_1, \dots, i_s$  from  $v_{i_j,0}$  to  $v_{0,2n}$ , then this costs at least  $2s$ . Note that this implies that any schedule  $S$  for delivering all messages by the agents such that every message is only carried by one agent satisfies  $\text{COST}(S) \geq 2r$ .

So let an agent  $a_{i,j}$  transport  $s$  messages from the source to the destination  $v_{0,2n}$ . Without loss of generality let these messages be  $1, \dots, s$ , which are picked up in this order. By construction, agent  $a_{i,j}$  needs to travel a distance of at least  $\frac{j}{n}$  to reach message 1, then distance 1 to move back to  $v_{0,n}$ , then distance 2 for picking up message  $i$  and going back to  $v_{0,n}$  for  $i = 2, \dots, s$ , and finally it needs to move distance 1 from  $v_{0,n}$  to  $v_{0,2n}$ . Overall, agent  $a_{i,j}$  therefore travels a distance of at least  $2s + \frac{j}{n}$ . The overall cost for agent  $a_{i,j}$  to deliver the  $s$  messages therefore is at least  $(2s + \frac{j}{n}) \cdot w_{i,j} = (2s + \frac{j}{n}) \cdot \frac{2r}{2r+j/n} \geq (2s + \frac{j}{n}) \cdot \frac{2s}{2s+j/n} = 2s$ .

Now, consider a schedule  $S_{\text{col}}$ , where the agents collaborate, i.e., agent  $a_{i,j}$  transports message  $i$  from  $v_{i,j}$  to  $v_{i,j+1}$  for  $i = 1, \dots, r$ ,  $j = 0, \dots, n-1$ , where we identify  $v_{i,n}$  with  $v_{0,n}$ . Then agent  $a_{0,j}$  transports all  $r$  messages from  $v_{0,j}$  to  $v_{0,j+1}$  for  $j = n, \dots, 2n-1$ . This is possible because  $r \leq \kappa$  by the choice of  $r$ . The total cost of this schedule is given by

$$\text{COST}(S_{\text{col}}) = r \cdot \int_0^1 f_{\text{step}}(x) dx + \int_1^2 f_{\text{step}}(x) dx,$$

where  $f_{\text{step}}(x)$  is a step-function defined on  $[0, 2]$  giving the current cost of transporting the message, i.e.,  $f_{\text{step}}(x) = \frac{2r}{2r+j/n}$  on the interval  $[j/n, (j+1)/n)$  for  $j = 0, \dots, 2n-1$ . The first integral corresponds to the first part of the schedule, where the  $r$  messages are transported separately and therefore the cost of transporting message  $i$  from  $v_{i,j}$  to  $v_{i,j+1}$  is exactly  $\int_{j/n}^{(j+1)/n} f_{\text{step}}(x) dx = \frac{1}{n} \cdot \frac{2r}{2r+j/n}$ . The second part of the schedule corresponds to the part, where all  $r$  messages are transported together by one agent at a time.

Observe that the function  $f(x) = 2r \cdot \frac{1}{2r-1/n+x}$  satisfies  $f(x) \geq f_{\text{step}}(x)$  on  $[0, 2]$ , hence

$$\begin{aligned} \text{COST}(S_{\text{col}}) &\leq r \int_0^1 f(x) dx + \int_1^2 f(x) dx = 2r \left( r \ln(2r - \frac{1}{n} + x) \Big|_0^1 + \ln(2r - \frac{1}{n} + x) \Big|_1^2 \right) \\ &= 2r \ln \left( \left( \frac{2r-1/n+1}{2r-1/n} \right)^r \left( \frac{2r-1/n+2}{2r-1/n+1} \right) \right) \xrightarrow{n \rightarrow \infty} 2r \ln \left( \left( 1 + \frac{1}{2r} \right)^r \left( 1 + \frac{1}{2r+1} \right) \right). \end{aligned}$$

Thus, the approximation ratio of an algorithm transporting every message by only one agent is bounded from below by  $\text{BoC} \geq \min_S \frac{\text{COST}(S)}{\text{COST}(S_{\text{col}})} \geq 1 / \ln \left( \left( 1 + \frac{1}{2r} \right)^r \left( 1 + \frac{1}{2r+1} \right) \right)$ . ◀

By observing that  $\lim_{r \rightarrow \infty} 1 / \ln \left( \left( 1 + 1/(2r) \right)^r \left( 1 + 1/(2r+1) \right) \right) = 1 / \ln(e^{1/2}) = 2$ , we obtain the following corollary.

► **Corollary 4.** *A schedule for WEIGHTEDDELIVERY where every message is delivered by a single agent cannot achieve an approximation ratio better than 2 in general, and better than  $1 / \ln 2 \approx 1.44$  for a single message.*

### 2.3 Upper Bounds on the Benefit of Collaboration

We now give tight upper bounds for Corollary 4. The following theorem shows that the benefit of collaboration is 2 in general. We remark that finding an optimal schedule in which every message is transported from its source to its destination by one agent, is already NP-hard, as shown in Theorem 8.



► **Theorem 5.** *Let  $\text{OPT}$  be an optimal schedule for a given instance of  $\text{WEIGHTEDDELIVERY}$ . Then there exists a schedule  $S$  such that every message is only transported by one agent and  $\text{COST}(S) \leq 2 \cdot \text{COST}(\text{OPT})$ .*

**Proof Sketch.** We may assume (without loss of generality) that the optimal schedule  $\text{OPT}$  transports each message along a simple path, and we construct a directed multigraph  $G_S$  with the same set of vertices and an arc for every move of an agent in  $\text{OPT}$ . We label every arc by the exact set of messages that were carried during the corresponding move in  $\text{OPT}$ . For every arc in  $G_S$  we add a backwards arc with the same label.

Obviously, every connected component of  $G_S$  is Eulerian, and we claim that any agent in each component can follow some Eulerian tour that allows to deliver all messages. In particular, the agent needs exactly twice as many moves as the total number of moves of all agents in the component in  $\text{OPT}$ . If we choose the cheapest agent (in terms of weight) in each component, we obtain a tour with at most twice the cost of  $\text{OPT}$ .

We compute the Eulerian for the cheapest agent of a component as a combination of multiple tours, respecting arc labels in the following sense: During every move along a forward arc, the agent carries the exact set of messages prescribed by the arc label, and during every move along a backward arc, the agent does not carry any messages. This ensures that all messages travel along the same path as in  $\text{OPT}$ . Whenever the agent is at a vertex  $v$  and is missing message  $i$  in order to proceed along some path, this means the current vertex must lie on  $i$ 's path in  $\text{OPT}$ , and thus there must be a path of backwards edges to the current location of  $i$ . The agent follows this path and recursively brings the message back to  $v$ . In the process, more recursive calls may be necessary, but we can prove that there cannot be a circular dependence between messages.

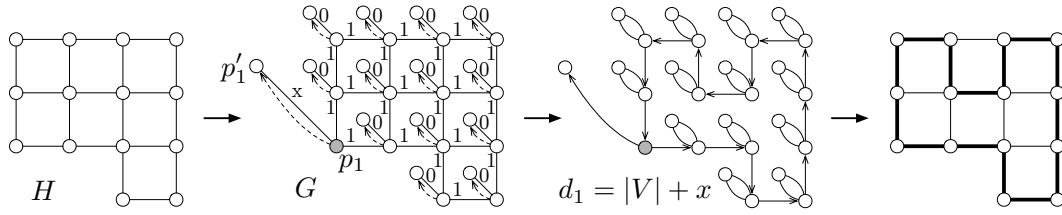
Therefore, the procedure eventually terminates after computing a closed tour. Note that, so far, the tour is still “virtual” in the sense that the agent didn’t actually move but merely computed the tour. We remove the tour from  $G_S$ , update all message positions, and recursively apply the procedure starting from the last vertex along the tour that is still adjacent to untraversed edges. By combining all (virtual) tours that we obtain in the recursion, we eventually get a Eulerian tour for the agent that obeys all arc labels. This means that the agent can successfully simulate all moves in  $\text{OPT}$  while ensuring that it is carrying the required messages before each move. ◀

**Single Message.** For the case of a single message, we can improve the upper bound of 2 on the benefit of collaboration from Theorem 5, to a tight bound of  $1/\ln 2 \approx 1.44$ .

The idea of the proof is to use that the weights are non-increasing by Lemma 1. After scaling appropriately, we assume the message path to be the interval  $[0, 1]$  and then choose a  $b$  such that the function  $\frac{b}{x+1}$  is a lower bound on the weight of the agent transporting the message at point  $x$  on the message path. The intersection of  $\frac{b}{x+1}$  and the step-function  $f$  representing the weight of the agent currently transporting the message then gives an agent, which can transport the message with at most  $(1/\ln 2)$ -times the cost of an optimal schedule.

► **Theorem 6.** *For  $\text{WEIGHTEDDELIVERY}$  with  $m = 1$ , there exists a  $(1/\ln 2)$ -approximation algorithm using a single agent.*

**No Intermediate Dropoffs.** For the capacities  $\kappa = 1$  and  $\kappa = \infty$ , the upper bound of 2 on the benefit of collaboration still holds if we additionally demand that each message is carried by its single agent without any intermediate dropoffs. We will make use of this result later in the approximation algorithm for  $\text{WEIGHTEDDELIVERY}$  with  $\kappa = 1$  (Section 5).



■ **Figure 3** Finding a Hamiltonian cycle via WEIGHTEDDELIVERY with a single agent. Picking  $x$  to be large enough, e.g.  $x = |V|$ , allows us to enforce that the agent will end in  $p'_1$ .

► **Theorem 7.** Let  $\text{OPT}$  be an optimal schedule for a given instance of WEIGHTEDDELIVERY with  $\kappa \in \{1, \infty\}$ . Then there exists a schedule  $S$  such that (i) every message is only transported by a single agent, with exactly one pick-up and one drop-off, (ii)  $\text{COST}(S) \leq 2 \cdot \text{COST}(\text{OPT})$ , and (iii) every agent  $a_j$  returns to its starting location  $p_j$ .

### 3 Planning

We now look in isolation at the problem of ordering the messages within the schedule of an agent, which we call *Planning*. Formally, the *Planning* aspect of WEIGHTEDDELIVERY is the following task: Given a schedule  $S$  and one of its agents  $a_j$ , reorder the actions in  $S|_{a_j}$  in such a way that the schedule remains feasible and the costs are minimized.

Generally speaking, for a complex schedule with many message handovers, the reordering options for a single agent  $a_j$  might be very limited. First of all, we must respect the capacity of  $a_j$ , i.e., in every prefix of  $S|_{a_j}$ , the number of pick-up actions  $(a_j, *, *, +)$  cannot exceed the number of drop-off actions  $(a_j, *, *, -)$  by more than  $\kappa$ . Even then, reordering  $S|_{a_j}$  might render  $S$  infeasible because of conflicts with some other subschedule  $S|_{a_x}$ . But *Planning* also includes the instances where a single agent delivers all the messages, one after the other straight to the target, and where the only thing that has to be decided is the ordering. We show now that in this setting, where there is no non-trivial *coordination* or *collaboration* aspect, WEIGHTEDDELIVERY is already NP-hard.

► **Theorem 8.** Planning of WEIGHTEDDELIVERY problem is NP-hard for all capacities  $\kappa$  even for a single agent on a planar graph.

The hardness follows by a reduction from Hamiltonian cycles on a grid graph  $H$ , a problem shown to be NP-hard by Itai et al. [21]: We put an isolated message at every node of  $H$ , forcing the agent to visit each node exactly once. A longer edge for the isolated message at the start forces the agent to come back to the start node towards the end, see Figure 3.

Using similar ideas, we can use recent results for the approximation hardness of metric TSP [22] to immediately show that *Planning* of WEIGHTEDDELIVERY can not be approximated arbitrarily well, unless  $\text{P} = \text{NP}$ .

► **Theorem 9.** It is NP-hard to approximate the Planning of WEIGHTEDDELIVERY to within any constant approximation ratio less than  $367/366$ .

#### 3.1 Polynomial-time Approximation for Planning in Restricted Settings

Motivated by Theorem 7, we now look at the restricted setting of planning for a feasible schedule  $S^R$  of which we know that each message is completely transported by some agent  $a_j$  without intermediate drop-offs, i.e., for every message  $m_i$  there must be an agent  $j$  with

$S^R|_{m_i} = (a_j, s_i, m_i, +), (a_j, t_i, m_i, -)$ . This allows us to give polynomial-time approximations for planning with capacity  $\kappa \in \{1, \infty\}$ :

► **Theorem 10.** *Let  $S^R$  be a feasible schedule for a given instance of WEIGHTEDDELIVERY with the restriction that  $\forall i \exists j : S^R|_{m_i} = (a_j, s_i, m_i, +), (a_j, t_i, m_i, -)$ . Denote by  $\text{OPT}(S^R)$  a reordering of  $S^R$  with optimal cost. There is a polynomial-time planning algorithm  $\text{ALG}$  which gives a reordering  $\text{ALG}(S^R)$  such that  $\text{COST}(\text{ALG}(S^R)) \leq 2 \cdot \text{COST}(\text{OPT}(S^R))$  if  $\kappa = 1$  and  $\text{COST}(\text{ALG}(S^R)) \leq 3.5 \cdot \text{COST}(\text{OPT}(S^R))$  if  $\kappa = \infty$ .*

**Proof.** By the given restriction, separate planning of each  $S^R|_{a_j}$  independently maintains feasibility of  $S^R$ . We denote by  $m_{j1}, m_{j2}, \dots, m_{jx}$  the messages appearing in  $S^R|_{a_j}$ . We define a complete undirected auxiliary graph  $G' = (V', E')$  on the node set  $V' = \{p_j\} \cup \{s_{j1}, s_{j2}, \dots, s_{jx}\} \cup \{t_{j1}, \dots, t_{jx}\}$  with edges  $(u, v)$  having length  $d_G(u, v)$ .

For  $\kappa = 1$ , the schedule  $\text{OPT}(S^R)|_{a_j}$  corresponds to a Hamiltonian path  $H$  in  $G'$  of minimum length, starting in  $p_j$ , subject to the condition that for each message  $m_{ji}$  the visit of its source  $s_{ji}$  is directly followed by a visit of its destination  $t_{ji}$ . We can lower bound the length of  $H$  with the total length of a spanning tree  $T' = (V', E(T')) \subseteq G'$  as follows: Starting with an empty graph on  $V'$  we first add all edges  $(s_{ji}, t_{ji})$ . Following the idea of Kruskal [23], we add edges from  $\{p_j\} \times \{s_{j1}, \dots, s_{jx}\} \cup \{t_{j1}, \dots, t_{jx}\} \times \{s_{j1}, \dots, s_{jx}\}$  in increasing order of their lengths, disregarding any edges which would result in the creation of a cycle. Now a DFS-traversal of  $T'$  starting from  $p_j$  visits any edge  $(s_{ji}, t_{ji})$  in both directions. Whenever we cross such an edge from  $s_{ji}$  to  $t_{ji}$ , we add  $(a_j, s_{ji}, m_{ji}, +), (a_j, t_{ji}, m_{ji}, -)$  as a suffix to the current schedule  $\text{ALG}(S^R)|_{a_j}$ . We get an overall cost of  $\text{COST}(\text{ALG}(S^R)|_{a_j}) \leq 2 \cdot \sum_{e \in E(T')} l_e \leq 2 \cdot \sum_{e \in H} l_e = 2 \cdot \text{COST}(\text{OPT}(S^R)|_{a_j})$ .

For  $\kappa = \infty$ , the idea is to first collect all messages by traversing a spanning tree (with cost  $\leq 2 \cdot \text{COST}(\text{OPT}(S^R)|_{a_j})$ ) and then delivering all of them in a metric TSP path fashion (with cost  $\leq \frac{3}{2} \cdot \text{COST}(\text{OPT}(S^R)|_{a_j})$ ). ◀

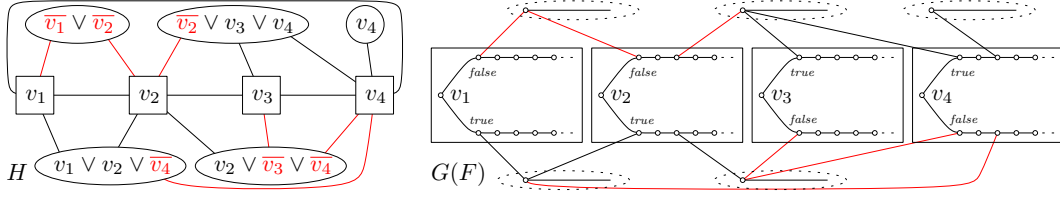
► **Remark.** If we assume as an additional property that the agent returns to its starting position  $p_j$  (as for example in the result of Theorem 7), we can get a better approximation for the case  $\kappa = 1$ . Instead of traversing a spanning tree twice, we can model this as the *stacker-crane problem* for which a polynomial-time 1.8-approximation is known [16].

## 4 Coordination

In this section, we focus on the *Coordination* aspect of WEIGHTEDDELIVERY. We assume that collaboration and planning are taken care of. More precisely, we are given a sequence containing the complete *fixed* schedule  $S^-$  of all actions  $(\_, s_i, m_i, +), \dots, (\_, h, m_i, -), \dots, (\_, t_j, m_j, -)$ , but without an assignment of the agents to the actions. Coordination is the task of assigning agents to the given actions. Even though coordination appears to have the flavor of a matching problem, it turns out to be NP-hard to optimally match up agents with the given actions. This holds for any capacity, in particular for  $\kappa = 1$ . The latter, however, has a polynomial-time solution if all agents have uniform weight.

### 4.1 NP-Hardness for Planar Graphs

We give a reduction from planar 3SAT: From a given planar 3SAT formula  $F$  we construct an instance of WEIGHTEDDELIVERY that allows a schedule  $S$  with “good” energy  $\text{COST}(S)$  if and only if  $F$  is satisfiable.



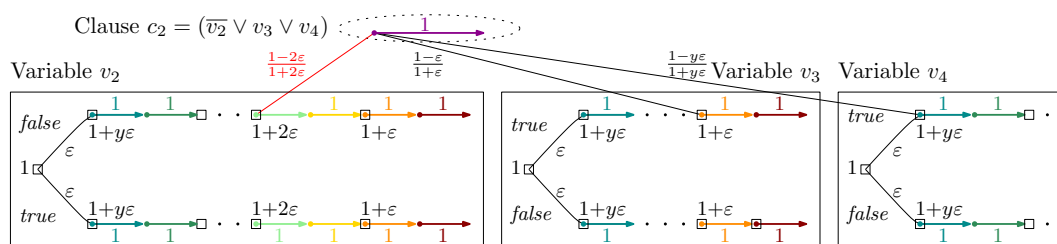
■ **Figure 4** (left) A restricted plane embedding of a 3CNF  $F$  which is satisfied by  $(v_1, v_2, v_3, v_4) = (true, false, false, true)$ . (right) Its transformation to the corresponding delivery graph.

**Planar 3SAT.** Let  $F$  be a three-conjunctive normal form (3CNF) with  $x$  boolean variables  $V(F) = \{v_1, \dots, v_x\}$  and  $y$  clauses  $C(F) = \{c_1, \dots, c_y\}$ . Each clause is given by a subset of at most three literals of the form  $l(v_i) \in \{v_i, \bar{v}_i\}$ . We define a corresponding graph  $H(F) = (N, A)$  with a node set consisting of all clauses and all variables ( $N = V(F) \cup C(F)$ ). We add an edge between a clause  $c$  and a variable  $v$ , if  $v$  or  $\bar{v}$  is contained in  $c$ . Furthermore we add a cycle consisting of edges between all pairs of consecutive variables, i.e.,  $A = A_1 \cup A_2$ , where  $A_1 = \{\{c_i, v_j\} \mid v_j \in c_i \text{ or } \bar{v}_j \in c_i\}$ ,  $A_2 = \{\{v_j, v_{(j \bmod x)+1}\} \mid 1 \leq j \leq x\}$ . We call  $F$  *planar* if there is a plane embedding of  $H(F)$ . The *planar 3SAT* problem of deciding whether a given planar 3CNF  $F$  is satisfiable is known to be NP-complete. Furthermore the problem remains NP-complete *if at each variable node the plane embedding is required to have all arcs representing positive literals on one side of the cycle  $A_2$  and all arcs representing negative literals on the other side of  $A_2$*  [26]. We will use this *restricted version* in our reduction and assume without loss of generality that the graph  $H(F) \setminus A_2$  is connected and that  $H(F)$  is a simple graph (i.e. each variable appears at most once in every clause).

**Building the Delivery Graph.** We first describe a way to transform any planar 3CNF graph  $H(F)$  into a planar delivery graph  $G = G(F)$ , see Figure 4.

We transform the graph in five steps: First we delete all edges of the cycle  $A_2$ , but we keep in mind that at each variable node all positive literal edges lie on one side and all negative literal edges on the other side. Secondly let  $\deg_{H(F), A_1}(v)$  denote the remaining degree of a variable node  $v$  in  $H$  and surround each variable node by a *variable box*. A variable box contains two paths adjacent to  $v$  on which internally we place  $\deg_{H(F), A_1}(v)$  copies of  $v$ : One path (called henceforth the *true-path*) contains all nodes having an adjacent positive literal edge, the other path (the *false-path*) contains all nodes having an adjacent negative literal edge. In a next step, we add a single node between any pair of node copies of the previous step. As a fourth step, we want all paths to contain the same number of nodes, hence we fill in nodes at the end of each path such that every path contains exactly  $2y \geq 2 \deg_{H(F), A_1}(v)$  internal nodes. Thus each variable box contains a variable node  $v$ , an adjacent *true-path* (with internal nodes  $v_{true,1}, \dots, v_{true,2y-1}$  and a final node  $v_{true,2y}$ ) and a respective *false-path*. Finally for each clause node  $c$  we add a new node  $c'$  which we connect to  $c$ . The new graph  $G(F)$  has polynomial size and all the steps can be implemented in such a way that  $G(F)$  is planar.

**Messages, Agents and Weights.** We are going to place one *clause message* on each of the  $y$  clause nodes and a *literal message* on each of the  $2x$  paths in the variable boxes for a total of  $4xy$  messages. More precisely, on each original clause node  $c$  we place exactly one clause message which has to be delivered to the newly created node  $c'$ . Furthermore we place a literal message on every internal node  $v_{true,i}$  of a *true-path* and set its target to  $v_{true,i+1}$  (same for the *false-path*). We set the length of all edges connecting a source to its target to 1.



■ **Figure 5** Agent positions ( $\square$ ) and weights (in black); Messages ( $\rightarrow$ ) and edge lengths (in color).

Next we describe the locations of the agents in each variable box. We place one *variable agent* of weight 1 on the variable node  $v$ . The length of the two adjacent edges are set to  $\varepsilon$ , where  $\varepsilon := (8xy)^{-2}$ . Furthermore we place  $y$  *literal agents* on each path: The  $i$ -th agent is placed on  $v_{true,2(y-i)}$  (respectively  $v_{false,2(y-i)}$ ) and gets weight  $1 + i\varepsilon$ . It remains to set the length of edges between clause nodes and internal nodes of a path. By construction the latter is the starting position of an agent of uniquely defined weight  $1 + i\varepsilon$ ; we set the length of the edge to  $\frac{1-i\varepsilon}{1+i\varepsilon}$ . For an illustration see Figure 5, where each agent's starting location is depicted by a square and each message is depicted by a colored arrow.

**Reduction.** The key idea of the reduction is that for each variable  $u$ , the corresponding variable box contains a variable agent who can *either* deliver all messages on the *true*-path (thus setting the variable to true), *or* deliver all messages on the *false*-path (thus setting the variable to false). Assume  $u$  is set to true. If  $u$  is contained in a clause  $c$ , then on the adjacent node  $v_{true,i}$  there is a (not yet used) literal agent. Intuitively, this agent was *freed by the variable agent* and can thus be sent to deliver the clause-message. If  $\bar{u}$  is contained in  $c$ , the corresponding literal on the *false*-path can't be sent to deliver the clause message, since it needs to transport messages along the *false*-path.

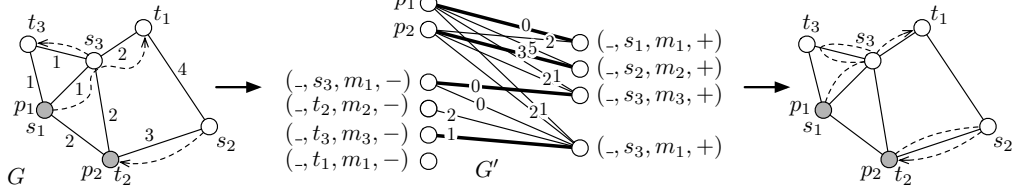
There is such a feasible schedule SOL of the agents in  $G(F)$  if and only if there is a satisfiable assignment (a solution) for the variables of a 3CNF  $F$ . Its total (energy) cost is  $\text{COST}(\text{SOL}) := 4xy + 2y + x(y^2 + y + 1)\varepsilon$  ([6, Lemma 12]). Furthermore, we can show that *any* schedule  $S$  which doesn't correspond to a satisfiable variable assignment has cost  $\text{COST}(S) > \text{COST}(\text{SOL})$  ([6, Lemma 13, 14 and 15]). This is true independent of whether  $S$  adheres to the schedule without agents  $S^-$  or not and holds for any capacity  $\kappa$ .

**Fixed Sequence (Schedule without Agent Assignment).** It remains to fix a sequence  $S^-$  that describes the schedule SOL described in the reduction idea but which does not allow us to infer a satisfiable assignment: This is the case for any  $S^-$  consisting of consecutive pairs  $(\_, s_i, m_i, +)$ ,  $(\_, t_i, m_i, -)$  such that if  $m_i$  lies to the left of  $m_j$  on some *true*- or *false*-path, it precedes  $m_j$  in the schedule.

► **Theorem 11.** *Coordination of WEIGHTEDDELIVERY is NP-hard on planar graphs for all capacities  $\kappa$ , even if we are given prescribed collaboration and planning.*

## 4.2 Polynomial-time Algorithm for Uniform Weights and Unit Capacity

Note that Coordination is NP-hard even for capacity  $\kappa = 1$ . Next we show that this setting is approachable once we restrict ourselves to uniform weights.



■ **Figure 6** Illustration of the coordination of the schedule  $S = (\_, s_1, m_1, +), (\_, s_2, m_2, +), (\_, s_3, m_1, -), (\_, s_3, m_3, +), (\_, t_2, m_2, -), (\_, t_3, m_3, -), (\_, s_3, m_1, +), (\_, t_1, m_1, -)$ . (left) Instance with 3 messages and 2 agents of uniform weight. (center) Equivalent weighted bipartite matching problem  $G'$ . (right) The resulting trajectories of the agents.

► **Theorem 12.** *Given collaboration and planning in the form of a complete schedule with missing agent assignment, Coordination of WEIGHTEDDELIVERY with capacity  $\kappa = 1$  and agents having uniform weights can be solved in polynomial time.*

**Proof.** As before, denote by  $S^- = (\_, s_i, m_i, +), \dots, (\_, h, m_i, -), \dots, (\_, t_j, m_j, -)$  the prescribed schedule without agent assignments. Since all agents have the same uniform weight  $w$ , the cost  $\text{COST}(S)$  of any feasible schedule  $S$  is determined by  $\text{COST}(S) = w \cdot \sum_{j=1}^k d_j$ . Hence at a pick-up action  $(\_, q, m_i, +)$  it is not so much important *which* agent picks up the message as *where / how far* it comes from.

Because we have capacity  $\kappa = 1$ , we know that the agent has to come from either its starting position or from a preceding drop-off action  $(\_, p, m_j, -) \in S^-$ . This allows us to model the problem as a weighted bipartite matching, see Figure 6 (center). We build an auxiliary graph  $G' = (A \cup B, E'_1 \cup E'_2)$ . A maximum matching in this bipartite graph will tell us for every pick-up action in  $B$ , where the agent that performs the pick-up action comes from in  $A$ . Let  $A := \{p_1, \dots, p_k\} \cup \{(\_, *, *, -)\}$  and  $B := \{(\_, *, *, +)\}$ . We add edges between all agent starting positions and all pick-ups,  $E'_1 := \{p_1, \dots, p_k\} \times \{(\_, q, m, +) \mid (\_, q, m, +) \in B\}$  of weight  $d_G(p_i, q)$ . Furthermore, we add edges between drop-offs and all subsequent pick-ups  $E'_2 := \{((\_, p, m_j, -), (\_, q, m_i, +)) \mid (\_, p, m_j, -) < (\_, q, m_i, +) \text{ in } S^-\}$  of weight  $d_G(p, q)$ .

A maximum matching of minimum cost in  $G'$  captures the optimal assignment of agents to messages and can be found by solving the classic *assignment problem*, a special case of the *minimum cost maximum flow problem*. Both of these problems can be solved in polynomial time for instance using the *Hungarian method* [24] or the *successive shortest path algorithm* [14], respectively. The cost of this optimum matching corresponds to the cost of the agents moving around without messages. The cost of the agents while carrying the messages can easily be added: Consider the schedule  $S^-$  restricted to a message  $m_i$ . This subsequence  $S^-|_{m_i}$  is a sequence of pairs of pick-up/drop-off actions  $((\_, q, m_i, +), (\_, p, m_i, -))$ , and in every pair the message is brought from  $q$  to  $p$  on the shortest path, so we add  $\sum d_G(q, p)$ . Concatenating these piecewise shortest paths gives the trajectory of each agent in the optimum solution, as illustrated in Figure 6 (right). ◀

Our algorithm is remotely inspired by a simpler problem at the ACM ICPC world finals 2015 [19]. The official solution is pseudo-polynomial [20], Austrin and Wojtaszczyk [3] later sketched a min-cost bipartite matching solution.

## 5 Approximation Algorithm

We have seen in Section 2 that the cost of an optimal schedule  $\text{OPT}$  for WEIGHTEDDELIVERY can be approximated to within a factor of two by restricting ourselves to schedules  $S^R$  in

which every message is only transported by a single agent, with exactly one pick-up and one drop-off (Theorem 7). Let  $\text{OPT}^R$  be an optimal restricted schedule. Given an auxiliary graph  $G'$  on a vertex set consisting of all agent positions and all message source and destination nodes, we construct in polynomial time a minimum tree cover of  $G'$  from which we build a schedule  $S^*$  in which agents travel at most twice the distance of their counterparts in  $\text{OPT}^R$  (similarly to Theorem 10). Here we require capacity  $\kappa = 1$ . In order for our method to work, we need to be indifferent between different weights; we achieve this by boosting each agent's weight  $w_j$  to  $\max w_i$ , resulting in an additional loss in the approximation factor of  $\max \frac{w_i}{w_j}$ :

► **Theorem 13.** *There is a polynomial-time  $(4 \max \frac{w_i}{w_j})$ -approximation algorithm for WEIGHTEDDELIVERY with capacity  $\kappa = 1$ .*

---

## References

- 1 Julian Anaya, Jérémie Chalopin, Jurek Czyzowicz, Arnaud Labourel, Andrzej Pelc, and Yann Vaxès. Convergecast and broadcast by power-aware mobile agents. *Algorithmica*, 74(1):117–155, 2016. doi:10.1007/s00453-014-9939-8.
- 2 David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA, 2007.
- 3 Per Austrin and Jakub Onufry Wojtaszczyk. ACM ICPC World Finals 2015 solution sketches. <http://www.csc.kth.se/~austrin/icpc/finals2015solutions.pdf>, 2015.
- 4 A. Bärtschi, J. Chalopin, S. Das, Y. Disser, B. Geissmann, D. Graf, A. Labourel, and M. Mihalák. Collaborative Delivery with Energy-Constrained Mobile Robots. In *23rd International Colloquium on Structural Information and Communication Complexity SIROCCO'16*, 2016.
- 5 A. Bärtschi, J. Chalopin, S. Das, Y. Disser, B. Geissmann, D. Graf, A. Labourel, and M. Mihalák. Collaborative Delivery with Energy-Constrained Mobile Robots, arXiv preprint, 2016. arXiv:1608.08500.
- 6 A. Bärtschi, J. Chalopin, S. Das, Y. Disser, D. Graf, J. Hackfeld, A. Labourel, and P. Penna. Energy-efficient Delivery by Heterogeneous Mobile Agents, arXiv preprint, 2016. URL: <https://arxiv.org/abs/1610.02361>.
- 7 Davide Bilò, Yann Disser, Luciano Gualà, Matús Mihalák, Guido Proietti, and Peter Widmayer. Polygon-constrained motion planning problems. In *9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics ALGOSENSORS'13*, pages 67–82, 2013. doi:10.1007/978-3-642-45346-5\_6.
- 8 Jérémie Chalopin, Shantanu Das, Matús Mihalák, Paolo Penna, and Peter Widmayer. Data delivery by energy-constrained mobile agents. In *9th International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics ALGOSENSORS'13*, pages 111–122, 2013. doi:10.1007/978-3-642-45346-5\_9.
- 9 Jérémie Chalopin, Riko Jacob, Matús Mihalák, and Peter Widmayer. Data delivery by energy-constrained mobile agents on a line. In *41st International Colloquium on Automata, Languages, and Programming ICALP'14*, pages 423–434, 2014. doi:10.1007/978-3-662-43951-7\_36.
- 10 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, February 1976.
- 11 Jurek Czyzowicz, Krzysztof Diks, Jean Moussi, and Wojciech Rytter. Communication problems for mobile agents exchanging energy. In *23rd International Colloquium on Structural Information and Communication Complexity SIROCCO'16*, 2016.

- 12 Erik D. Demaine, Mohammadtaghi Hajiaghayi, Hamid Mahini, Amin S. Sayedi-Roshkhar, Shayan Oveisgharan, and Morteza Zadimoghaddam. Minimizing movement. *ACM Transactions on Algorithms (TALG)*, 5(3):1–30, 2009. doi:10.1145/1541885.1541891.
- 13 Jack Edmonds and Ellis L. Johnson. Matching, euler tours and the chinese postman. *Mathematical Programming*, 5(1):88–124, 1973. doi:10.1007/BF01580113.
- 14 Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.
- 15 Robert W. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.
- 16 P. Fraigniaud, L. Gasieniec, D. Kowalski, and A. Pelc. Collective tree exploration. In *6th Latin American Theoretical Informatics Symposium LATIN'04*, pages 141–151, 2004.
- 17 Greg N. Frederickson, Matthew S. Hecht, and Chul E. Kim. Approximation algorithms for some routing problems. In *Proceedings of the 17th Annual Symposium on Foundations of Computer Science (FOCS)*, 1976.
- 18 J. A. Hoogeveen. Analysis of christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10(5):291–295, July 1991. doi:10.1016/0167-6377(91)90016-I.
- 19 ACM ICPC. World Finals 2015 Problems, Task C: Catering. <https://icpc.baylor.edu/worldfinals/problems>, May 2015.
- 20 ICPCNews. ACM ICPC 2015 problem catering. [https://www.youtube.com/watch?v=WxCgcI18\\_d8](https://www.youtube.com/watch?v=WxCgcI18_d8), May 2015.
- 21 Alon Itai, Christos H Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.
- 22 Marek Karpinski, Michael Lampis, and Richard Schmied. New inapproximability bounds for tsp. *Journal of Computer and System Sciences*, 81(8):1665–1677, 2015.
- 23 Joseph B. Kruskal, Jr. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.
- 24 Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.
- 25 Chung-Lun Li, S. Thomas McCormick, and David Simchi-Levi. The point-to-point delivery and connection problems: Complexity and algorithms. *Discrete Applied Mathematics*, 36(3):267–292, 1992.
- 26 David Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982. doi:10.1137/0211025.
- 27 Stephen Warshall. A theorem on boolean matrices. *Journal of the ACM (JACM)*, 9(1):11–12, 1962.
- 28 Pawel Winter. Steiner problem in networks: A survey. *Networks*, 17(2):129–167, 1987. doi:10.1002/net.3230170203.



# Towards Tighter Space Bounds for Counting Triangles and Other Substructures in Graph Streams<sup>\*†</sup>

Suman K. Bera<sup>1</sup> and Amit Chakrabarti<sup>2</sup>

<sup>1</sup> Department of Computer Science, Dartmouth College, Hanover, USA

<sup>2</sup> Department of Computer Science, Dartmouth College, Hanover, USA

---

## Abstract

We revisit the much-studied problem of space-efficiently estimating the number of triangles in a graph stream, and extensions of this problem to counting fixed-sized cliques and cycles. For the important special case of counting triangles, we give a 4-pass,  $(1 \pm \varepsilon)$ -approximate, randomized algorithm using  $\tilde{O}(\varepsilon^{-2} m^{3/2}/T)$  space, where  $m$  is the number of edges and  $T$  is a promised lower bound on the number of triangles. This matches the space bound of a recent algorithm (McGregor et al., PODS 2016), with an arguably simpler and more general technique. We give an improved multi-pass lower bound of  $\Omega(\min\{m^{3/2}/T, m/\sqrt{T}\})$ , applicable at essentially all densities  $\Omega(n) \leq m \leq O(n^2)$ . We prove other multi-pass lower bounds in terms of various structural parameters of the input graph. Together, our results resolve a couple of open questions raised in recent work (Braverman et al., ICALP 2013).

Our presentation emphasizes more general frameworks, for both upper and lower bounds. We give a sampling algorithm for counting arbitrary subgraphs and then improve it via combinatorial means in the special cases of counting odd cliques and odd cycles. Our results show that these problems are considerably easier in the cash-register streaming model than in the turnstile model, where previous work had focused (Manjunath et al., ESA 2011; Kane et al., ICALP 2012). We use Turán graphs and related gadgets to derive lower bounds for counting cliques and cycles, with triangle-counting lower bounds following as a corollary.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

**Keywords and phrases** data streaming, graph algorithms, triangles, subgraph counting, lower bounds

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.11

## 1 Introduction

Algorithms for analyzing large graphs have been the topic of two decades of intense research. The theory of such algorithms encompasses two large disciplines: *streaming* algorithms [25, 14], where the input graph is presented as a stream of edges that must be read sequentially, in one or more passes, using space sublinear in the total input size; and *property-testing* algorithms [16, 17], where the input graph may be randomly accessed and the goal is to decide whether it satisfies some property or is far from doing so, while reading a sublinear fraction of the input. This paper is concerned with streaming algorithms.

---

\* In this extended abstract, several proofs are either shortened or omitted, and a few theorems and lemmas are stated in not-quite-full detail. We refer the interested reader to the full version of this paper.

† This work was supported in part by the NSF under Award 1650992.



■ **Table 1** Results from prior work. See the discussion at the start of Section 1.1.

Problem	Space	Remarks	Source	
Triangle counting (TRI-CNT)	$\tilde{O}(mn/T)^2$	–	[3]	
	$\tilde{O}(m\Delta^2/T)$	$\Delta =$ maximum degree	[19]	
	$\tilde{O}(mn/T)$	$n$ known a priori	[8]	
	$\tilde{O}(m^3/T^2)$	turnstile	[21]	
	$\tilde{O}(m\Delta/T)$	$\Delta =$ maximum degree	[28]	
	$\tilde{O}(m\gamma/T)$	$\gamma =$ tangle coefficient (Section 2)	[28]	
	$\tilde{O}(mJ/T + m/\sqrt{T})$	$J =$ max # triangles containing an edge	[27]	
	$C + \tilde{O}(P_2/T)$	$C =$ vertex cover, $P_2 =$ # of 2-paths	[15]	
	$\tilde{O}(m/\sqrt{T})$	dependence on $\varepsilon$ is $1/\varepsilon^{2.5}$	[11]	
	$\tilde{O}(m^{3/2}/T)$	multi-pass	[24]	
	$\tilde{O}(m/\sqrt{T})$	multi-pass	[24]	
		$\Omega(n^2)$	one pass, $T = 1$	[3]
		$\Omega(n/T)$	multi-pass, $T < n$	[19]
		$\Omega(m)$	one pass, $m \in [c_1n, c_2n^2]$ , $T < n$	[7]
	$\Omega(m/T)$	multi-pass	[7]	
	$\Omega(m^3/T^2)$	one pass, optimal	[22]	
	$\Omega(m/T^{2/3})$	multi-pass	[11]	
	$\Omega(m/\sqrt{T})$	multi-pass, for $m = \Theta(n\sqrt{T})$	[11]	
Clique counting (CLQ-CNT <sub><i>h</i></sub> )	$\tilde{O}(m^{h(h-1)/2}/T^2)$	turnstile	[21]	
	$\tilde{O}(\eta(h)/T)$	$\eta(h) = \max\{m^\alpha \Delta^{h-2\alpha} : \alpha \in \{1, \lfloor h/2 \rfloor\}\}$	[28]	
Cycle counting (CYC-CNT <sub><i>h</i></sub> )	$\tilde{O}(m^h/T^2)$	turnstile	[23]	

Specifically, this paper is about the SUBGRAPH-COUNTING problem, which asks for an (approximate) count of the number of occurrences of a particular constant-sized subgraph,  $H$ , in an input graph,  $G$ , which has  $n$  vertices and  $m$  edges. We denote this problem SUB-CNT <sub>$H$</sub> . After giving a basic algorithm for SUB-CNT <sub>$H$</sub> , we provide improvements (in terms of space usage) for special classes of subgraphs, namely, when  $H$  is either an  $h$ -clique  $\mathcal{K}_h$  (the CLIQUE-COUNTING problem, or CLQ-CNT <sub>$h$</sub> ) or an  $h$ -cycle  $\mathcal{C}_h$  (the CYCLE-COUNTING problem, or CYC-CNT <sub>$h$</sub> ). We also give upper and lower bounds, several of them optimal, for the very important special case of TRIANGLE-COUNTING (henceforth, TRI-CNT), when  $H$  is a triangle.

The number of triangles in a graph is a basic parameter of interest for a variety of reasons, including social network analysis [26] and spam and fraud detection [4]; see [12, 31] for a more thorough discussion of applications. The TRI-CNT problem has been the focus of a remarkably large number of papers. Nevertheless, the definitive upper and lower bounds on its space complexity have not yet been obtained, leaving us with several mutually incomparable results (Table 1). Notably, distinguishing a triangle-free graph from a graph containing one or more triangles requires  $\Omega(n^2)$  space in general [3], ruling out unconditional sublinear-space solutions. Therefore, nontrivial bounds must either assume some structural guarantees on the input graph or provide space guarantees that depend on some structural parameter.

## 1.1 Our Results and Comparison with Prior Work

To set the context for our results, we summarize the salient related results from prior work (most of which are about TRI-CNT) in Table 1. The ubiquitous parameter “ $T$ ” represents a

■ **Table 2** Our main results. The  $\tilde{O}$ -notation hides  $1/\varepsilon^2$  and  $\log m$  factors;  $h$  is considered a constant. The upper bounds use 4 passes for counting odd cliques and odd cycles (including triangles) and 2 passes in all other cases. The notations  $\Delta$ ,  $\gamma$ ,  $\rho$ , and  $\beta$  are as discussed at the start of Section 1.1.

Problem	Upper bounds	Lower bounds		Remarks
	multi-pass	one pass	multi-pass	
TRI-CNT	$\tilde{O}(m^{3/2}/T)$	–	$\Omega\left(\min\{m^{3/2}/T, m/\sqrt{T}\}\right)$ $\Omega(m\Delta/T)$ $\Omega(m\gamma/T)$ $\Omega(m/\rho)$	optimal optimal optimal optimal
CLQ-CNT <sub>h</sub>	$\tilde{O}(m^{h/2}/T)$	$\Omega(m^h/T^2)$	$\Omega\left(\min\{m^{h/2}/T, m/T^{1/(h-1)}\}\right)$	
CYC-CNT <sub>h</sub>	$\tilde{O}(m^{h/2}/T)$	$\Omega(m^{h/2}/T)$	$\Omega(m^{h/2}/T)$	even $h$
	$\tilde{O}(m^{h/2}/T)$	$\Omega(m^h/T^2)$	$\Omega\left(\min\{m^{h/2}/T, m/T^{1/(h-1)}\}\right)$	odd $h$
SUB-CNT <sub>H</sub>	$\tilde{O}(m^{\beta(H)}/T)$	–	–	

guaranteed lower bound on the number of copies of the target subgraph (triangle,  $\mathcal{K}_h$ , or  $\mathcal{C}_h$ ) in the input graph  $G$ . Some of the results involve additional graph parameters (definitions in Section 2): the maximum degree  $\Delta = \Delta(G)$ , the triangle density  $\rho = \rho(G)$ , the tangle coefficient  $\gamma = \gamma(G)$  and, for the problem SUB-CNT<sub>H</sub>, the edge cover number  $\beta = \beta(H)$ .

In reading Table 1, note that all upper bounds represent randomized algorithms that provide  $(1 \pm \varepsilon)$ -multiplicative approximations with high constant probability, and work in one pass, except as noted. Their space usage involves a factor of  $\log m$  and an  $\varepsilon$ -dependent factor that, with one exception, equals  $1/\varepsilon^2$ . We omit these factors for clarity, hiding them into an  $\tilde{O}(\cdot)$  notation. Lower bounds are proven by studying the communication complexity of distinguishing between “high” and “low” values of  $T$ . As can be seen, there are not many multi-pass lower bounds in prior work and none that explain the form of any of the upper bounds. Our lower bounds in this paper serve to fill this explanatory gap. Meanwhile, our upper bounds demonstrate the power of using a small constant number of passes, rather than one pass, for these problems. Table 2 summarizes our main contributions.

Our first upper bound is for SUB-CNT<sub>H</sub>. We give a 2-pass algorithm based on (implicitly) sampling a suitable set of vertices and then counting copies of  $H$  induced by this set. Since the edge cover number,  $\beta(H)$ , equals  $\lceil h/2 \rceil$  when  $H$  is either  $\mathcal{K}_h$  or  $\mathcal{C}_h$ , our general upper bound for SUB-CNT<sub>H</sub> already implies the claimed upper bounds for CLQ-CNT<sub>h</sub> and CYC-CNT<sub>h</sub> for even  $h$ . To improve these bounds in the case of odd  $h$  – in particular, TRI-CNT – we bring in some combinatorial ideas from Eden et al. [13], who were interested in a query complexity version of TRI-CNT; an overview of these ideas appears at the start of Section 3.2. We believe that the resulting algorithms are conceptually novel in a streaming context.

Braverman et al. [7] introduced the parameter *triangle density*,  $\rho$ , defining it to be the number of vertices that belong to some triangle in  $G$ . They conjectured a lower bound of  $\Omega(m/\rho)$  for multi-pass TRIANGLE-COUNTING. Our Theorem 4.12 settles this conjecture positively. Further, they posed the problem of designing a multi-pass algorithm for TRIANGLE-COUNTING with space complexity depending only on  $m$  and  $T$  (presumably they meant one that beats the one-pass upper bound of Kane et al. [21]). We address this in Algorithm 2. Pavan et al. [28] introduced the *tangle coefficient*,  $\gamma$ ; see Section 2. Our results in Theorem 4.13 and Corollary 4.14 show that each of their one-pass upper bounds – namely,  $\tilde{O}(m\gamma/T)$  and  $\tilde{O}(m\Delta/T)$  – is matched by a multi-pass lower bound.

Jha et al. [18] gave an  $O(m/\sqrt{T})$ -space algorithm for a variant of TRI-CNT where the error guarantee is additive; they also showed how to multiplicatively estimate a related quantity called the *clustering coefficient*. While related, these results are not directly comparable to ours, or to the ones in Table 1. Very recently, McGregor et al. [24] extended some ideas from this work of Jha et al., obtaining an  $\tilde{O}(m/\sqrt{T})$ -space algorithm for TRI-CNT. In the same paper, they also gave an  $\tilde{O}(m^{3/2}/T)$ -space 4-pass<sup>1</sup> algorithm for TRI-CNT, which matches the space bound of our algorithm for TRI-CNT (Corollary 3.9). However, their algorithm relies on a more complex primitive of fast  $\ell_p$  sampling and is not immediately generalizable to counting larger subgraphs. Our algorithm, which uses only basic sampling, is arguably simpler, solves a more general problem, and is space-optimal for counting cliques and odd cycles in some parameter regimes.

On the lower bound side, at a high level we proceed along the expected lines of reducing from the INDEX and SET-DISJOINTNESS communication problems, for one-pass and multi-pass bounds, respectively. The meat of the work is in designing appropriate gadgets, mostly based on Turán graphs, for the reductions. The most closely-related work is by Cormode and Jowhari [11], who give multi-pass  $\Omega(m/T^{2/3})$  and  $\Omega(m/\sqrt{T})$  lower bounds<sup>2</sup> for TRI-CNT. Their constructions imply these lower bounds for specific settings of the edge-density and number-of-triangles parameters. Our own lower bounds for TRI-CNT (Corollary 4.11) apply at all edge densities between  $m = \Theta(n)$  and  $m = \Theta(n^2)$  and at all triangle counts between  $T = 1$  and  $T = m^{3/2-\delta}$ . Moreover, our bounds generalize to CLQ-CNT and CYC-CNT.

## 2 Preliminaries

Throughout this paper, our input graph will be  $G = (V, E)$ , a simple undirected graph with  $|V| = n$  and  $|E| = m$ . For a vertex  $v \in V$ ,  $N_v$  denotes its set of neighbors and  $d_v = |N_v|$  denotes its degree. We put  $\Delta = \max_{v \in V} d_v$ . The input graph is presented as a stream of edges  $(e_1, e_2, \dots, e_m)$  in some adversarial order. Each edge in  $E$  appears exactly once and the stream only builds up the graph: there are no edge deletions. This is sometimes called the cash-register streaming model.

For an edge  $e \in E$ , slightly abusing notation,  $N_e$  denotes the set of edges in  $E$  that are adjacent to  $e$ . Let  $N_e^>$  be the set of edges in  $N_e$  that come after  $e$  in the streaming order; thus  $N_e^> \subseteq N_e$ . Let  $\mathcal{T}$  denote the set of triangles in  $G$ . We now define some special graph parameters that are useful in the context of subgraph counting algorithms: the first two were introduced in the context of the TRI-CNT problem.

► **Definition 2.1.** The *triangle density* [7]  $\rho = \rho(G)$  is the number of vertices that belong to some triangle in  $G$ . With  $T = |\mathcal{T}|$ , we have  $\Theta(T^{1/3}) \leq \rho \leq 3T$ , where the lower and upper bounds correspond to a clique and  $T$  vertex-disjoint triangles, respectively.

► **Definition 2.2.** Suppose  $T = |\mathcal{T}| > 0$ . For  $\tau \in \mathcal{T}$ , let  $e(\tau)$  be the edge in  $\tau$  appearing earliest in the stream order. The *tangle coefficient* [28] of the stream presentation of  $G$ , denoted  $\gamma = \gamma(G)$ , is defined as  $\gamma = (1/T) \sum_{\tau \in \mathcal{T}} |N_{e(\tau)}^>|$ . Clearly,  $\gamma \leq 2\Delta$ , since  $|N_{e(\tau)}^>| \leq 2\Delta$ .

► **Definition 2.3.** An edge cover of a graph is a set of edges that covers all the vertices. For a graph  $H$ , its *edge cover number*, denoted  $\beta(H)$ , is the cardinality of its smallest edge cover.

<sup>1</sup> McGregor et al. state their result as a 3-pass algorithm in a nonstandard model where vertex degrees can be queried for free. In this model, our algorithm for TRI-CNT would also use only 3 passes.

<sup>2</sup> The Cormode–Jowhari lower bound of  $\Omega(m/\sqrt{T})$ , does not contradict our upper bound of  $\tilde{O}(m^{3/2}/T)$  because their lower bound holds only at  $m = \Theta(n\sqrt{T})$  and  $T \leq n^2$  triangles.

We fix a total ordering on the vertices of  $G$  according to their degrees. For vertices  $u$  and  $v$ , let  $u \prec v$  if either  $d_u < d_v$ , or  $d_u = d_v$  and  $u < v$  lexicographically. We record an important observation made in Eden et al. [13].

► **Fact 2.4** (Eden et al. [13]). *For each  $u \in V$ ,  $|\{v \in N_u : u \prec v\}| \leq \sqrt{2m}$ .*

Let  $Q$  be some nonnegative function of an input stream that we wish to estimate. Let  $\varepsilon, \delta \in [0, 1]$  be certain parameters. If an algorithm  $\mathcal{A}$  produces an estimate  $\hat{Q}$  for  $Q$  such that  $\Pr[\hat{Q} \in (1 \pm \varepsilon)Q] \geq 1 - \delta$ , then  $\mathcal{A}$  is called an  $(\varepsilon, \delta)$ -estimator for  $Q$ . Our algorithms will follow the common strategy of designing an unbiased “basic estimator” for  $Q$  – i.e., a random variable with expectation  $Q$  – and bounding its variance. We note the following widely-used lemma that combines several such basic estimators (computed in parallel) into an  $(\varepsilon, \delta)$ -estimator.

► **Lemma 2.5** (Median-of-Means Improvement [2, 9]). *Let  $X$  be the distribution of an unbiased estimator for a real quantity  $Q$ . Let  $\{X_{ij}\}_{i \in [t], j \in [k]}$  be a collection of i.i.d. copies of  $X$ , where  $t = c \log(1/\delta)$  and  $k = 3 \text{Var}[X]/(\varepsilon^2 \mathbb{E}[X]^2)$ , for a certain universal positive constant  $c$ . Let  $Z = \text{median}_{i \in [t]}(\frac{1}{k} \sum_{j=1}^k X_{ij})$ . Then  $\Pr[Z \in (1 \pm \varepsilon)Q] \geq 1 - \delta$ .*

### 3 Algorithms for Counting Subgraphs

In this section we present multi-pass algorithms for  $\text{SUB-CNT}_H$ , the problem of estimating the number of occurrences of a fixed subgraph  $H$  of constant order. We first consider general  $H$  and give a 2-pass algorithm. When specialized to  $\mathcal{K}_h$  and  $\mathcal{C}_h$ , this algorithm uses  $\tilde{O}(m^{\lceil h/2 \rceil}/T)$  space. Later, for the case of odd  $h = 2\ell + 1$ , we introduce additional combinatorial ideas to improve the exponent of  $m$  from  $\ell + 1$  to  $\ell + \frac{1}{2}$ , at the cost of two additional passes. In particular, this gives us an  $\tilde{O}(m^{3/2}/T)$ -space  $\text{TRI-CNT}$  algorithm.

#### 3.1 A Sampling-Based Algorithm for Arbitrary Subgraphs

► **Theorem 3.1.** *Let  $H$  be a graph of constant order whose edge cover number is  $\beta$ . There is an  $(\varepsilon, 1/3)$ -estimator for  $\text{SUB-CNT}_H$  that uses two passes and  $\tilde{O}(S)$  space, provided  $S = \Omega(m^\beta/T)$ , where  $T$  is the number of distinct occurrences of  $H$  in the input graph.*

► **Remark.** The above bound could instead have been stated as  $\tilde{O}(m^\beta/T)$  with  $T$  being a promised lower bound on the number of distinct occurrences of  $H$ . Similar remarks apply to our other upper bounds. We remind the reader that  $\tilde{O}(\cdot)$  hides  $1/\varepsilon^2$  and  $\log m$  factors.

**Proof.** Let  $V(H)$  and  $E(H)$  be the vertex set and edge set of  $H$ , respectively. Let  $\xi$  be the number of *lists* of distinct edges of  $H$  that form minimum-sized edge covers of  $H$ .<sup>3</sup> Note that  $\beta$  and  $\xi$  are constants, independent of the input graph  $G$ . Therefore the following algorithm, which reads a stream of the  $m$  edges of  $G$  and computes an estimator  $X$ , uses  $\tilde{O}(1)$  space.

The analysis of Algorithm 1 is handled by the next two lemmas, which show that  $X$  is an unbiased estimator and that its variance can be controlled. Let  $H_1, H_2, \dots, H_T$  be the occurrences of  $H$  in  $G$ . Let  $T_i$  be an indicator random variable to denote whether  $H_i$  is detected in Pass 2, at Line 4. Then  $X = \frac{m^\beta}{\xi} \sum_{i=1}^T T_i$ .

► **Lemma 3.2.** *For each  $i$ ,  $\mathbb{E}[T_i] = \xi/m^\beta$ . Thus,  $\mathbb{E}[X] = T$ .*

<sup>3</sup> E.g., if  $H$  is  $\mathcal{C}_4$ , with edges  $a, b, c, d$  in cyclic order, these lists are  $(a, c)$ ,  $(b, d)$ ,  $(c, a)$ , and  $(d, b)$ ; so  $\xi = 4$ .

---

**Algorithm 1** Basic estimator for SUB-CNT<sub>H</sub>.
 

---

**Pass 1:**

- 1: select  $\beta$  edges  $\{e_i = \{u_i, v_i\}\}_{i=1}^\beta$  independently and u.a.r., using reservoir sampling
- 2: **if**  $\{u_1, v_1, \dots, u_\beta, v_\beta\}$  does not have exactly  $|V(H)|$  distinct vertices **then**
- 3:      $X \leftarrow 0$ ; abort

**Pass 2:**

- 4:    $c \leftarrow$  number of distinct copies of  $H$  on  $\{u_1, v_1, \dots, u_\beta, v_\beta\}$  that contain each of  $e_1, \dots, e_\beta$
  - 5:    $X \leftarrow cm^\beta/\xi$
- 

► **Lemma 3.3.**  $\text{Var}[X] = O(m^\beta T)$ .

**Proof.** By Lemma 3.2,

$$\text{Var}[X] \leq \mathbb{E}[X^2] = \frac{m^{2\beta}}{\xi^2} \left( \sum_{i=1}^T \mathbb{E}[T_i^2] + \sum_{i \neq j} \mathbb{E}[T_i T_j] \right) = \frac{m^{2\beta}}{\xi^2} \left( \frac{T\xi}{m^\beta} + \sum_{i \neq j} \mathbb{E}[T_i T_j] \right). \quad (1)$$

The term  $\mathbb{E}[T_i T_j]$ , with  $i \neq j$ , is nonzero iff  $H_i$  and  $H_j$  can be simultaneously detected at Line 4. Examining the logic of the algorithm, we see that this can happen only if  $V(H_i) = V(H_j)$  and there is a set of  $\beta$  edges that forms a minimum edge cover of both  $H_i$  and  $H_j$ . When these conditions hold, we shall say that  $H_i$  and  $H_j$  are *neighbors*. Since  $H$  is a constant-order graph, each  $H_i$  has  $O(1)$  many neighbors: a crude bound, but one that suffices for our purposes.

Thus, in the double summation in (1), only  $O(T)$  terms are nonzero. For each nonzero term, we have  $\mathbb{E}[T_i T_j] = \Pr[T_i = 1 \wedge T_j = 1] \leq \Pr[T_i = 1] = \frac{\xi}{m^\beta}$ . Plugging this into (1), we obtain our required estimate. ◀

To complete the proof of Theorem 3.1, we invoke Lemma 2.5 on the unbiased estimator  $X$  and use the above bound on  $\text{Var}[X]$ . ◀

Let us specialize the above result to the problems CLQ-CNT<sub>h</sub> and CYC-CNT<sub>h</sub>. We have  $\beta(\mathcal{K}_h) = \beta(\mathcal{C}_h) = \lceil h/2 \rceil$ . Therefore, Theorem 3.1 gives us a 2-pass estimator that uses  $\tilde{O}(S)$  space, provided  $S = \Omega(m^{\lceil h/2 \rceil}/T)$ , where  $T$  is the number of  $h$ -cliques or  $h$ -cycles (as appropriate) in the input graph. We shall later show, in Theorem 4.2, that this bound is optimal for CLQ-CNT<sub>h</sub> when  $h$  is even.

### 3.2 An Improved Algorithm for Odd Cliques and Odd Cycles

We now present an algorithm for CLQ-CNT<sub>2 $\ell$ +1</sub>, for constant  $\ell$ , improving the space bound from  $\tilde{O}(m^{\ell+1}/T)$ , as implied by Theorem 3.1, to  $\tilde{O}(m^{\ell+\frac{1}{2}}/T)$ . As before, all space bounds are for an  $(\varepsilon, 1/3)$ -estimator.

Our algorithm builds on ideas from Pavan et al. [28] and Eden et al. [13]. The former paper gives a streaming algorithm for estimating the number of triangles  $T$  in a graph. The idea is to sample an edge uniformly at random from the stream, using reservoir sampling; then sample one more edge uniformly at random from the neighborhood of previously chosen edge; and finally, check whether these two edges are closed by any edge in the “future stream” to form a triangle. This leads to an unbiased estimator for  $T$  with variance bounded by  $O(m\Delta T)$ . This leads to an  $\tilde{O}(m\Delta/T)$ -space  $(\varepsilon, 1/3)$ -estimator, with the caveat that prior

knowledge of  $\Delta$  is required. We build on their algorithm by improving the bound on the variance of the unbiased estimator to  $O(m^{3/2}T)$ . This gives an  $\tilde{O}(m^{3/2}/T)$  space estimator for TRI-CNT as well as removes the dependency on  $\Delta$ . We reduce the variance of our estimator by repeating the neighborhood sampling step for edges whose endpoints have “large” degree.

The challenge now is to reduce the number of triangles that an edge participates in. For this, we use an idea of vertex ordering from Eden et al. [13], who tackled triangle counting in a property testing model. They fix a total ordering on the vertices of  $G$  according to their degrees. For vertices  $u$  and  $v$ , let  $u \prec v$  if either  $d_u < d_v$ , or  $d_u = d_v$  and  $u < v$  lexicographically. Now let  $\tau = \{v_1, v_2, v_3\}$  be a triangle in  $G$  with  $v_1 \prec v_2 \prec v_3$ . Then  $\tau$  can be associated with  $(e_1, v_3)$  where  $e_1 = \{v_1, v_2\}$ . Observe that each triangle  $\tau$  in  $G$  is uniquely associated with a distinct edge. Let the number of triangles associated with edge  $e$  be  $T_e$ . Clearly,  $\sum_{e \in E(G)} T_e = T$ . From Fact 2.4, it follows that  $T_e \leq \sqrt{2m}$ . Since each edge is associated with not-too-many triangles, we get a “strong” upper bound on the variance. In fact the idea of vertex ordering has been proved to be useful for counting triangles in other (offline) settings as well [30]. By invoking a result from Chiba et al. [10], we show that in spite of such repetition the space usage for our estimator remains constant in expectation. We in fact show that we can generalize this idea for larger cliques.

Now we formally describe our estimator for CLQ-CNT. We fix a total ordering of  $V(G)$  as described above. Let  $\tau = \{v_1, v_2, \dots, v_{2\ell}, v_{2\ell+1}\}$  induce a  $\mathcal{K}_{2\ell+1}$  in  $G$  with  $v_1 \prec v_2 \prec \dots \prec v_{2\ell} \prec v_{2\ell+1}$ . We associate  $\tau$  with  $(e_1, e_2, \dots, e_\ell, v_{2\ell+1})$  where  $e_i = \{v_{2i-1}, v_{2i}\}$ . Observe that each  $\mathcal{K}_{2\ell+1}$  in  $G$  is uniquely associated with  $\ell$  distinct edges. Let the number of  $(2\ell+1)$ -cliques associated with  $(e_1, e_2, \dots, e_\ell)$  be  $T_{(e_1, e_2, \dots, e_\ell)}$ . Then, we have the following simple lemma.

► **Lemma 3.4.** *Each  $T_{(e_1, e_2, \dots, e_\ell)} \leq \sqrt{2m}$ . Further,  $\sum_{(e_1, e_2, \dots, e_\ell) \in E(G)^\ell} T_{(e_1, e_2, \dots, e_\ell)} = T$ .*

We shall also need the following combinatorial lemma, from Chiba and Nishizeki [10].

► **Lemma 3.5** (Based on Lemmas 1(a) and 2 of [10]). *Let  $G = (V, E)$  be a graph with  $n$  vertices and  $m$  edges such that  $m = \Omega(n)$ . Then  $\sum_{\{u, v\} \in E} \min\{d_u, d_v\} = O(m^{3/2})$ .*

Algorithm 2 computes our basic estimator  $X$ .

► **Theorem 3.6.** *Suppose the input graph  $G$  contains  $T$  copies of  $\mathcal{K}_{2\ell+1}$ . Then Algorithm 2 leads to an  $\tilde{O}(S)$ -space  $(\varepsilon, 1/3)$ -estimator for  $T$  when  $S = \Omega(m^{\ell+1/2}/T)$ .<sup>4</sup>*

**Proof.** Let  $\mathcal{E}_{(e_1, e_2, \dots, e_\ell)}$  be the event that edges  $e_1 = \{u_1, v_1\}, e_2 = \{u_2, v_2\}, \dots, e_\ell = \{u_\ell, v_\ell\}$  are sampled at Line 1 and the algorithm does not abort in Pass 1. WLOG assume  $u_i \prec v_i$  for all  $i \in [\ell]$ . In the next two lemmas, we shall show that  $X$  is an unbiased estimator and its variance can be controlled. Then we shall analyze the space usage of Algorithm 2.

► **Lemma 3.7.** *For each  $\mathcal{E}_{(e_1, e_2, \dots, e_\ell)}$ ,  $\mathbb{E}[Z_k \mid \mathcal{E}_{(e_1, e_2, \dots, e_\ell)}] = T_{(e_1, e_2, \dots, e_\ell)}$ . Thus,  $\mathbb{E}[X] = T$ .*

► **Lemma 3.8.**  $\text{Var}[X] = O(m^{\ell+1/2}T)$ .

**Proof.** As in the previous lemma,  $\mathbb{E}[Z_k^2 \mid \mathcal{E}_{(e_1, e_2, \dots, e_\ell)}] = d_{u_1} T_{(e_1, e_2, \dots, e_\ell)}$ . Next, note that  $Z_{k_1}$  and  $Z_{k_2}$  are independent for  $k_1 \neq k_2$ , even after conditioning on  $\mathcal{E}_{(e_1, e_2, \dots, e_\ell)}$ . Therefore

$$\mathbb{E}[Z_{k_1} Z_{k_2} \mid \mathcal{E}_{(e_1, e_2, \dots, e_\ell)}] = \mathbb{E}[Z_{k_1} \mid \mathcal{E}_{(e_1, e_2, \dots, e_\ell)}] \cdot \mathbb{E}[Z_{k_2} \mid \mathcal{E}_{(e_1, e_2, \dots, e_\ell)}] = T_{(e_1, e_2, \dots, e_\ell)}^2. \quad (2)$$

<sup>4</sup> Please see the remark after Theorem 3.1 for an alternate interpretation of the space bound.

---

**Algorithm 2** Basic estimator for  $\text{CLQ-CNT}_{2\ell+1}$ .

---

**Pass 1:**

- 1: select  $\ell$  edges  $\{e_i = \{u_i, v_i\}\}_{i=1}^\ell$  independently and u.a.r., using reservoir sampling
- 2: **if**  $\{u_1, v_1, \dots, u_\ell, v_\ell\}$  does not have exactly  $2\ell$  distinct vertices **then**
- 3:      $X \leftarrow 0$ ; abort

**Pass 2:**

- 4: count  $d_{u_i}$  and  $d_{v_i}$  for all  $i \in [\ell]$

**Pass 3:**

- 5:  $r \leftarrow \lceil \min\{d_{u_1}, d_{v_1}\} / \sqrt{m} \rceil$
- 6: by renaming vertices if needed, ensure that  $u_i \prec v_i$  for all  $i \in [\ell]$
- 7: **for**  $k \leftarrow 1$  **to**  $r$  **do**
- 8:      $Z_k \leftarrow 0$ ; select a vertex  $w_k$  from  $N_{u_1}$  u.a.r., using reservoir sampling

**Pass 4:**

- 9: compute  $d_{w_1}, \dots, d_{w_r}$
- 10: **for**  $k \leftarrow 1$  **to**  $r$  **do**
- 11:     **if**  $(e_1, \dots, e_\ell, w_k)$  forms a  $\mathcal{K}_{2\ell+1}$  and  $u_1 \prec v_1 \prec u_2 \prec \dots \prec u_\ell \prec v_\ell \prec w_k$  **then**
- 12:          $Z_k \leftarrow d_{u_1}$
- 13:  $Y \leftarrow (1/r) \sum_{k=1}^r Z_k$
- 14:  $X \leftarrow m^\ell Y$

---

Now, using Lemma 3.7,

$$\begin{aligned} \mathbb{E}[Y^2 \mid \mathcal{E}_{(e_1, e_2, \dots, e_\ell)}] &= \frac{1}{r^2} \mathbb{E} \left[ \left( \sum_{k=1}^r Z_k \right)^2 \mid \mathcal{E}_{(e_1, e_2, \dots, e_\ell)} \right] \\ &= \frac{1}{r^2} \sum_{k=1}^r \mathbb{E}[Z_k^2 \mid \mathcal{E}_{(e_1, e_2, \dots, e_\ell)}] + \frac{1}{r^2} \sum_{k_1 \neq k_2} \mathbb{E}[Z_{k_1} Z_{k_2} \mid \mathcal{E}_{(e_1, e_2, \dots, e_\ell)}] \\ &\leq \frac{d_{u_1}}{r} T_{(e_1, e_2, \dots, e_\ell)} + T_{(e_1, e_2, \dots, e_\ell)}^2 = \sqrt{m} T_{(e_1, e_2, \dots, e_\ell)} + T_{(e_1, e_2, \dots, e_\ell)}^2. \end{aligned}$$

Therefore,

$$\begin{aligned} \text{Var}[X] &= m^{2\ell} \text{Var}[Y] \leq m^{2\ell} \mathbb{E}[Y^2] = m^{2\ell} \cdot \frac{1}{m^\ell} \sum_{(e_1, e_2, \dots, e_\ell) \in E^\ell} \mathbb{E}[Y^2 \mid \mathcal{E}_{(e_1, e_2, \dots, e_\ell)}] \\ &\leq m^{\ell+1/2} \sum_{(e_1, \dots, e_\ell) \in E^\ell} T_{(e_1, \dots, e_\ell)} + m^\ell \sum_{(e_1, \dots, e_\ell) \in E^\ell} T_{(e_1, \dots, e_\ell)}^2 = (1 + \sqrt{2}) m^{\ell+1/2} T, \end{aligned}$$

where the final step uses both parts of Lemma 3.4.  $\blacktriangleleft$

We return to the proof of Theorem 3.6. Invoking Lemma 2.5, we get an  $(\varepsilon, 1/3)$ -approximation algorithm for  $T$  with space  $\tilde{O}(m^{\ell+1/2} \cdot B/T)$  bits, where  $B$  is the space used by Algorithm 2. To estimate  $B$ , note that Algorithm 2 keeps  $r = \lceil \min\{d_{u_1}, d_{v_1}\} / \sqrt{m} \rceil$  many neighboring vertices of  $u_1$ . Recall that the edge  $\{u_1, v_1\}$  is chosen uniformly at random. Hence the expected space usage is

$$\frac{1}{m} \sum_{\{u, v\} \in E(G)} \left\lceil \frac{\min\{d_u, d_v\}}{\sqrt{m}} \right\rceil \leq 1 + \frac{1}{m} \sum_{\{u, v\} \in E(G)} \frac{\min\{d_u, d_v\}}{\sqrt{m}} = O(1),$$

where the final step uses Lemma 3.5.



We can easily turn this expected space bound – call it  $B_0$  – into a worst-case space bound that holds w.h.p. We simply abort the algorithm if we find that  $r$  would exceed  $10B_0$  (say). By Markov’s inequality, this ensures that  $B \leq 10B_0 = O(1)$ , with probability at least  $\frac{9}{10}$ .

Thus, the overall space usage of our final estimator is  $\tilde{O}(m^{\ell+\frac{1}{2}}/T)$ , as required. ◀

Given the importance of the problem of counting triangles, it is worthwhile to record the following immediate corollary of Theorem 3.6.

► **Corollary 3.9.** *For a graph with  $n$  vertices,  $m$  edges, and  $T$  triangles, the TRI-CNT problem admits a four-pass  $\tilde{O}(S)$ -space  $(\varepsilon, 1/3)$ -estimator when  $S = \Omega(m^{3/2}/T)$ .*

A similar improvement is possible for counting odd cycles of constant order. Details, including a pseudocode description of the relevant algorithm, appear in the full paper.

► **Theorem 3.10.** *For a graph with  $m$  edges, and  $T$  copies of  $\mathcal{C}_{2\ell+1}$ , the CYC-CNT $_{2\ell+1}$  problem admits a four-pass  $\tilde{O}(S)$ -space  $(\varepsilon, 1/3)$ -estimator when  $S = \Omega(m^{\ell+\frac{1}{2}}/T)$ .*

## 4 Lower Bounds

The remainder of this paper is concerned with lower bounds. Unless otherwise stated, all the lower bounds are for randomized algorithms with success probabilities at least  $2/3$ . We first prove multi-pass lower bounds for CLQ-CNT and CYC-CNT. The latter turns out to require different constructions for the cases of odd cycles and even cycles. In particular, this makes the lower bound for CYC-CNT $_h$  more restrictive when  $h$  is even. Next, we give single pass lower bounds for CLQ-CNT and CYC-CNT: our lower bound for even cycles is weaker than that for odd cycles. Finally, we focus on TRI-CNT and establish tight space lower bounds in terms of special structural parameters of the input graph, these parameters being ones studied in previous works on triangle counting.

For some intuition on why even cycles are harder to deal with, we recall an important result from extremal graph theory [6, 5]: for each integer  $\ell \geq 2$ ,  $\text{ex}(n, \mathcal{C}_{2\ell}) = O(\ell n^{1+1/\ell})$ , where  $\text{ex}(n, H) = \max\{m : \exists \text{ graph on } n \text{ vertices and } m \text{ edges that does not contain } H\}$ . Our lower bounds for CLQ-CNT $_h$  and CYC-CNT $_{2\ell+1}$  depend on constructing dense  $\mathcal{K}_h$ -free and  $\mathcal{C}_{2\ell+1}$ -free graphs (respectively). However, in view of the above bound, dense  $\mathcal{C}_{2\ell}$ -free graphs do not exist, necessitating weaker constructions.

### 4.1 Multi-Pass Lower Bounds

The source of these lower bounds is the following promise version of the SET-DISJOINTNESS problem that is of central importance in communication complexity theory. Alice and Bob have two  $N$ -bit strings  $x$  and  $y$  respectively, each with exactly  $R$  ones. They want to decide whether there exists an index  $i \in [N]$  such that  $x_i = 1 = y_i$ . Let this problem be denoted as DISJ $_N^R$ . The randomized communication complexity  $\text{R}(\text{DISJ}_N^R) = \Omega(R)$  for all  $R \leq N/2$  [20, 29].

Our reductions from DISJ $_N^R$  naturally lead to lower bounds on a clique *detection* problem, where the goal is to distinguish input graphs  $G$  that have no  $h$ -cliques from ones that have “many”  $h$ -cliques. Note that this provides a legitimate counterpart to our upper bounds, all of which will safely report “0” on input graphs that are  $h$ -clique-free.

► **Definition 4.1** (Clique and cycle detection problems). Consider two graph families:  $\mathcal{G}_1$  consisting of  $h$ -clique-free graphs with  $n$  vertices and  $m$  edges;  $\mathcal{G}_2$  consisting of graphs with

## 11:10 Counting Triangles and Other Substructures in Graph Streams

$n$  vertices,  $m$  edges, and at least  $T$   $h$ -cliques. Given a streamed input graph  $G \in \mathcal{G}_1 \cup \mathcal{G}_2$ ,  $\text{CLQ-DETECT}_h(p, n, m, T)$  is the problem of deciding whether  $G \in \mathcal{G}_1$  or  $G \in \mathcal{G}_2$  with success probability at least  $2/3$ , using at most  $p$  passes over the edge stream. Analogously, we define the cycle detection problem  $\text{CYC-DETECT}_h(p, n, m, T)$ .

We shall prove the following two theorems about clique detection.

► **Theorem 4.2.** *For each constant  $h$  and constant  $p$ , solving  $\text{CLQ-DETECT}_h(p, n, m, T)$  requires  $\Omega(m^{h/2}/T)$  bits of space, provided  $T = \Omega(m^{(h-1)/2})$ . Furthermore, this holds for any  $m = \Theta(n^r)$ ,  $1 \leq r \leq 2$ , and any  $T$  with  $\Omega(m^{(h-1)/2}) \leq T \leq m^{(h-1)/2+\delta}$ , where  $\delta$  is an arbitrary constant in  $[0, 1/2)$ .*

► **Theorem 4.3.** *For each constant  $h$ , constant  $p$ , and for any  $T$  with  $1 \leq T \leq m^{(h-1)/2}$ , solving  $\text{CLQ-DETECT}_h(p, n, m, T)$  requires  $\Omega(m/T^{1/(h-1)})$  bits of space.*

► **Remark.** Our lower bounds admit the possibility that there may exist more efficient algorithms when the number of cliques is relatively small in the graph.

We note that McGregor et al. [24] have given two algorithms that match our lower bounds when the subgraph of interest is a triangle (3-clique).

We present a detailed proof of Theorem 4.2 via reductions from  $\text{DISJ}_N^{N/3}$ ; for the proof of Theorem 4.3, see the full paper. Turán graphs play a central role. Recall that a Turán graph is a complete multipartite graph where the blocks (of the vertex partition) are as close as possible to being equal in size. Let  $T(n, t)$  denote an  $n$ -vertex  $t$ -partite Turán graph: it has  $t - (n \bmod t)$  blocks with  $\lfloor n/t \rfloor$  vertices each and another  $(n \bmod t)$  blocks with  $\lceil n/t \rceil$  vertices each. As is well known,  $T(n, t)$  is the densest  $n$ -vertex graph that is  $\mathcal{K}_{t+1}$ -free.

**Proof of Theorem 4.2.** We reduce  $\text{DISJ}_N^{N/3}$  to  $\text{CLQ-DETECT}_h(p, n, m, T)$  by constructing a certain graph that has some fixed edges, some edges depending on Alice's input,  $x$ , and some edges depending on Bob's input,  $y$ .

Let  $H = (V_H, E_H)$  be a copy of the Turán graph  $T(b(h-1), h-1)$ , with  $B_j$  denoting the  $j$ th block in  $V_H$ . By construction,  $|B_j| = b$  for all  $j \in [h-1]$ , and

$$E_H = \bigcup_{i \neq j} \{\{u, v\} : u \in B_i, v \in B_j\}.$$

To this fixed graph  $H$ , we add  $N$  additional blocks of vertices, denoted  $V_1, \dots, V_N$ , with each  $|V_i| = d$ . Then Alice and Bob add edge sets  $E_{\text{Alice}}$  and  $E_{\text{Bob}}$  respectively, defined as follows

$$E_{\text{Alice}} = \bigcup_{\substack{i: x_i=1, \\ j \in [h-2]}} \{\{u, v\} : u \in V_i, v \in B_j\}, \quad E_{\text{Bob}} = \bigcup_{j: y_j=1} \{\{u, v\} : u \in V_j, v \in B_{h-1}\}.$$

In words, for each index  $i$  with  $x_i = 1$ , Alice constructs a complete bipartite subgraph between  $V_i$  and  $B_j$  for all  $j \in [h-2]$ . Similarly, for each index  $j$  with  $y_j = 1$ , Bob creates a complete bipartite subgraph between  $V_j$  and  $B_{h-1}$ . Let the final resulting graph be denoted  $G_{\text{clique}} = (V_{\text{clique}}, E_{\text{clique}})$  where  $V_{\text{clique}} = V_H \cup (V_1 \cup \dots \cup V_N)$ , and  $E_{\text{clique}} = E_H \cup E_{\text{Alice}} \cup E_{\text{Bob}}$ .

► **Lemma 4.4.** *The graph  $G_{\text{clique}}$  is  $\mathcal{K}_h$ -free iff  $x$  and  $y$  are disjoint.*

**Proof.** Observe that graphs  $G_A = (V_{\text{clique}}, E_H \cup E_{\text{Alice}})$  and  $G_B = (V_{\text{clique}}, E_H \cup E_{\text{Bob}})$  are both  $\mathcal{K}_h$ -free. Thus, any  $h$ -clique in  $G_{\text{clique}}$  must be of the form  $\{v_1, \dots, v_h\}$  where  $v_1 \in B_1, \dots, v_{h-1} \in B_{h-1}$ , and  $v_h \in V_i$  for some  $i \in [N]$ . But this implies that  $V_i$  is connected to  $B_j$  for all  $j \in [h-1]$ . Hence,  $x_i = y_i = 1$ . ◀

Next, we record some important parameters of  $G_{\text{clique}}$ . First,  $|E_H| = \binom{h-1}{2}b^2 = \Theta(b^2)$ . Put  $n = |V_{\text{clique}}|$ ,  $m = |E_{\text{clique}}|$ . Let  $T_H$  denote the number of  $h$ -cliques in the graph. Recall that, as an instance of  $\text{DISJ}_N^{N/3}$ , each of  $x$  and  $y$  has exactly  $N/3$  ones. Thus,

$$n = |V_H| + \sum_{i=1}^N |V_i| = \Theta(b) + Nd, \quad m = |E_H| + |E_{\text{Alice}}| + |E_{\text{Bob}}| = \Theta(b^2) + \Theta(Nbd),$$

$$T_H = 0, \quad \text{if } x \cap y = \emptyset; \quad T_H \geq b^{h-1}d, \quad \text{otherwise.}$$

Setting  $b = N$  and  $d = 1$ , we get  $n = \Theta(N)$ ,  $m = \Theta(N^2)$ , and  $T_H \geq N^{h-1}$  if  $x$  and  $y$  are not disjoint. Suppose that there is an algorithm  $\mathcal{A}$  that solves  $\text{CLQ-DETECT}_h(p, n, m, T)$  with  $T = N^{h-1}$  in only  $o(m^{h/2}/T)$  space, for some constant  $p$ . Then there exists a communication protocol with cost  $o(R)$  that solves  $\text{DISJ}_N^{N/3}$ . This gives the main result of Theorem 4.2.

The proof so far applies to a graph with  $m = \Theta(n^2)$ . To generalize it to arbitrary  $m$  and  $T$ , assume  $m = \Theta(n^r)$ , and  $T = m^{(h-1)/2+\delta}$  for some fixed  $r, \delta$  such that  $1 \leq r \leq 2$ , and  $0 \leq \delta < 1/2$ . We modify the construction of  $G_{\text{clique}}$  as follows (note that  $\delta = 1/2$  gives the maximum possible number of  $h$ -cliques in a graph with  $m$  edges). We set  $b = N^q$  and  $d = N^{q-1}$  where  $q = 1/(\frac{r}{2} - \delta r)$ . Now mark  $b^{r/2}$  vertices in each block  $B_i$  and  $d^{\frac{qr-2}{2(q-1)}}$  vertices in each set  $V_i$  as *active* vertices. Then we only add edges between active vertices of each block. In the modified  $G_{\text{clique}}$ , we have

$$n = \Theta(N^q), \quad m = \Theta(b^r) + \Theta(Nb^{\frac{r}{2}}d^{\frac{qr-2}{2(q-1)}}) = \Theta(N^{qr}),$$

$$T_H = 0, \quad \text{if } x \cap y = \emptyset; \quad T_H \geq b^{(h-1)\frac{r}{2}}d^{\frac{qr-2}{2(q-1)}} = \Theta\left(N^{\frac{hqr}{2}-1}\right), \quad \text{otherwise.}$$

Plugging in  $q = 1/(\frac{r}{2} - \delta r)$ , we get  $T_H = \Theta(m^{(h-1)/2+\delta})$  when  $x$  and  $y$  are not disjoint. The lower bound of  $\Omega(N)$  for  $\text{DISJ}_N^{N/3}$  implies a lower bound of  $\Omega(m^{h/2}/T)$  for  $\text{CLQ-DETECT}_h(p, n, m, T)$  with  $T = m^{(h-1)/2+\delta}$ .  $\blacktriangleleft$

In the full paper, we prove the following lower bounds for  $\text{CYC-DETECT}_h(p, n, m, T)$ . The first two, for odd  $h$ , are analogous to Theorem 4.2 and Theorem 4.3, except that Turán graphs are replaced with an appropriate dense gadget. The third lower bound, for even  $h$ , uses a different, “sparse” gadget, leading to a more restrictive lower bound.

► **Theorem 4.5.** *For each odd constant  $h$  and constant  $p$ , solving  $\text{CYC-DETECT}_h(p, n, m, T)$  requires  $\Omega(m^{h/2}/T)$  bits of space, provided  $T = \Omega(m^{(h-1)/2})$ . Furthermore, this holds for any  $m = \Theta(n^r)$ ,  $1 \leq r \leq 2$ , and any  $T$  with  $\Omega(m^{(h-1)/2}) \leq T \leq m^{(h-1)/2+\delta}$ , where  $\delta$  is an arbitrary constant in  $[0, 1/2)$ .*

► **Theorem 4.6.** *For each odd constant  $h$ , constant  $p$ , and for any  $T$  with  $1 \leq T \leq m^{(h-1)/2}$ , solving  $\text{CYC-DETECT}_h(p, n, m, T)$  requires  $\Omega(m/T^{1/(h-1)})$  bits of space.*

► **Theorem 4.7.** *For each even constant  $h$  and constant  $p$ , there is a family of instances with  $m = \Theta(n)$  and  $T = \Theta(m^{(h-2)/2})$ , such that solving  $\text{CYC-DETECT}_h(p, n, m, T)$  requires  $\Omega(m^{h/2}/T)$  bits of space.*

## 4.2 Single Pass Lower Bounds

We also obtain one-pass streaming lower bounds for the special subgraph counting problems studied in the previous section. Proofs appear in the full paper. These bounds use reductions from the  $\text{INDEX}_N$  communication problem: Alice has a  $N$ -bit string,  $x$ , and Bob has an

index  $z \in [N]$ . The goal is to output the bit  $x_z$ . The one-way randomized communication complexity  $R^\rightarrow(\text{INDEX}_N) = \Omega(N)$  [1].

Lower bounds for CLQ-CNT and CYC-CNT are obtained by studying the corresponding detection problems CLQ-DETECT and CYC-DETECT. As before, CYC-DETECT $_h$  is treated differently for odd  $h$  and even  $h$ . In each of these theorems,  $h$  is a constant.

Theorem 4.8 and Theorem 4.10 have the weakness that they apply only at carefully chosen parameter settings, à la Theorem 4.7. A more thorough treatment of these theorems is deferred to the full paper.

► **Theorem 4.8.** *Solving CLQ-DETECT $_h(1, n, m, T)$  requires  $\Omega(m^{h-\varepsilon}/T^2)$  space for every small constant  $\varepsilon > 0$ .*

► **Theorem 4.9.** *For odd  $h$ , solving CYC-DETECT $_h(1, n, m, T)$  requires  $\Omega(m^h/T^2)$  space.*

► **Theorem 4.10.** *For even  $h$ , CYC-DETECT $_h(1, n, m, T)$  requires  $\Omega(m^{h/2}/T)$  space.*

### 4.3 Special Lower Bounds for Triangle Counting

Finally, we present some tight multi-pass space lower bounds for TRI-CNT in terms of graph structural parameters introduced in previous works. Analogous to Definition 4.1, we define TRI-DETECT( $p, n, m, T$ ), where the goal is to distinguish between graphs from “no triangles” family and “at least  $T$  triangles” family. TRI-DETECT-DENSITY( $p, n, m, T, \rho$ ), TRI-DETECT-TANGLE( $p, n, m, T, \gamma$ ), and TRI-DETECT-DEGREE( $p, n, m, T, \Delta$ ) are variants of this problem where the triangle density  $\rho$ , the tangle coefficient  $\gamma$ , and maximum degree  $\Delta$  (respectively) are supplied as parameters.

In each of the following theorems, the number of passes,  $p$ , is a constant.

As a direct consequence of Theorem 4.2 and Theorem 4.3, we have the following basic lower bound.

► **Corollary 4.11.** *Solving TRI-DETECT( $p, n, m, T$ ) requires  $\Omega\left(\min\{m/T^{2/3}, m/\sqrt{T}\}\right)$  space.*

We can prove the following lower bounds for other variants of TRI-DETECT by reductions from DISJ $_N^R$  using suitable gadgets. Details appear in the full paper.

► **Theorem 4.12.** *Solving TRI-DETECT-DENSITY( $p, n, m, T, \rho$ ) requires  $\Omega(m/\rho)$  space.*

► **Theorem 4.13.** *Solving TRI-DETECT-TANGLE( $p, n, m, T, \gamma$ ) requires  $\Omega(m\gamma/T)$  space.*

► **Theorem 4.14.** *Solving TRI-DETECT-DEGREE( $p, n, m, T, \Delta$ ) requires  $\Omega(m\Delta/T)$  space.*

## 5 Concluding Remarks

In this paper, we have made several advances in our understanding of the space complexity of subgraph counting problems. Nevertheless, a number of key problems remain open and we end by highlighting some significant ones.

- Consider the data streaming problems CLQ-CNT $_h$  (for arbitrary constant  $h$ ) and CYC-CNT $_h$  (for odd constant  $h$ ), using a constant number of passes. In each case, we have given a space lower bound of  $\Omega\left(\min\{m^{h/2}/T, m/T^{1/(h-1)}\}\right)$  and an upper bound of  $\tilde{O}(m^{h/2}/T)$ . Suppose that  $T$ , the actual number of cliques or cycles (as applicable) in the input graph, is relatively small: to be precise, suppose that  $T \leq m^{(h-1)/2}$ . In this regime, there is a gap between the upper and lower bounds, as discussed after Theorem 4.3. Can we design a constant-pass algorithm using  $\tilde{O}(m/T^{1/(h-1)})$  space?

- We have proved a one-pass lower bound of  $\Omega(m^h/T^2)$  for  $\text{CLQ-DETECT}_h$ . The best known one-pass upper bound for  $\text{CLQ-CNT}_h$  is  $\tilde{O}(m^{h(h-1)/2}/T^2)$  [21]. Bridging this gap remains an open problem. The situation for cycle counting is better: the upper bound of  $\tilde{O}(m^h/T^2)$  for  $\text{CYC-CNT}_h$  [23] matches our lower bound up to a logarithmic factor, when  $h$  is odd.
- Can one improve the one-pass and multi-pass lower bounds for  $\text{CYC-CNT}_h$  for even  $h$  to match those for odd  $h$ ? Since it is impossible to construct a “dense” graph without creating even cycles, one may hope that there exist more efficient algorithms for counting even cycles. It would be very interesting to settle the problem either way.

---

## References

- 1 Farid Ablayev. Lower bounds for one-way probabilistic communication complexity and their application to space complexity. *Theor. Comput. Sci.*, 175(2):139–159, 1996.
- 2 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. Preliminary version in *Proc. 28th Annual ACM Symposium on the Theory of Computing*, pages 20–29, 1996.
- 3 Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 623–632, 2002.
- 4 Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24–27, 2008*, pages 16–24, 2008.
- 5 Béla Bollobás. *Extremal Graph Theory*. Academic Press, New York, NY, 1978.
- 6 John A. Bondy and Miklós Simonovits. Cycles of even length in graphs. *Journal of Combinatorial Theory, Series B*, pages 97–105, 1974.
- 7 Vladimir Braverman, Rafail Ostrovsky, and Dan Vilenchik. How hard is counting triangles in the streaming model? In *Proc. 40th International Colloquium on Automata, Languages and Programming*, pages 244–254, 2013.
- 8 Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *Proc. 25th ACM Symposium on Principles of Database Systems*, pages 253–262, 2006.
- 9 Amit Chakrabarti. CS49: Data Stream Algorithms Lecture Notes, Fall 2011. URL: <http://www.cs.dartmouth.edu/~ac/Teach/data-streams-lecnotes.pdf>.
- 10 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.
- 11 Graham Cormode and Hossein Jowhari. A second look at counting triangles in graph streams. *Theoretical Computer Science*, 552:44–51, 2014.
- 12 David Easley and Jon Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, 2012.
- 13 Talya Eden, Amit Levi, Dana Ron, and C. Seshadhri. Approximately counting triangles in sublinear time. In *Proc. 56th Annual IEEE Symposium on Foundations of Computer Science*, pages 614–633, 2015.
- 14 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2–3):207–216, 2005. Preliminary version in *Proc. 31st International Colloquium on Automata, Languages and Programming*, pages 531–543, 2004.
- 15 David Garcia-Soriano and Konstantin Kutzkov. Triangle counting in streamed graphs via small vertex covers. *Tc*, 2:3, 2014.

- 16 Oded Goldreich. A brief introduction to property testing. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation – In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, pages 465–469. Springer, 2011. doi:10.1007/978-3-642-22670-0\_31.
- 17 Oded Goldreich. Introduction to testing graph properties. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation – In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, pages 470–506. Springer, 2011. doi:10.1007/978-3-642-22670-0\_32.
- 18 Madhav Jha, C. Seshadhri, and Ali Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *Proc. 19th Annual SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 589–597, 2013.
- 19 Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In *Computing and Combinatorics*, pages 710–716. Springer, 2005.
- 20 Bala Kalyanasundaram and Georg Schintger. The probabilistic communication complexity of set intersection. *SIAM J. Disc. Math.*, 5(4):545–557, 1992.
- 21 Daniel M Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In *Proc. 39th International Colloquium on Automata, Languages and Programming*, pages 598–609, 2012.
- 22 Konstantin Kutzkov and Rasmus Pagh. Triangle counting in dynamic graph streams. In *Proc. 14th Scandinavian Symposium and Workshops on Algorithm Theory*, pages 306–318, 2014.
- 23 Madhusudan Manjunath, Kurt Mehlhorn, Konstantinos Panagiotou, and He Sun. Approximate counting of cycles in streams. In *Proc. 19th Annual European Symposium on Algorithms*, pages 677–688, 2011.
- 24 Andrew McGregor, Sofya Vorotnikova, and Hoa T. Vu. Better algorithms for counting triangles in data streams. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 401–411, 2016.
- 25 S. Muthukrishnan. Data streams: Algorithms and applications. *Found. Trends Theor. Comput. Sci.*, 1(2):117–236, 2005. doi:10.1561/0400000002.
- 26 Mark E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003. doi:10.1137/S003614450342480.
- 27 Rasmus Pagh and Charalampos E. Tsourakakis. Colorful triangle counting and a mapreduce implementation. *Information Processing Letters*, 112(7):277–281, 2012.
- 28 Aduri Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Counting and sampling triangles from a graph stream. *Proceedings of the VLDB Endowment*, 6(14):1870–1881, 2013.
- 29 Alexander A. Razborov. On the distributional complexity of disjointness. *Theor. Comput. Sci.*, 106(2):385–390, 1992.
- 30 Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. In *Proceedings of the 20th international conference on World wide web*, pages 607–614, 2011.
- 31 Stanley Wasserman and Katherine Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

# On Polynomial Approximations Over $\mathbb{Z}/2^k\mathbb{Z}^{*\dagger}$

Abhishek Bhrushundi<sup>1</sup>, Prahladh Harsha<sup>2</sup>, and  
Srikanth Srinivasan<sup>3</sup>

- 1 Department of Computer Science, Rutgers University, New Brunswick, USA  
abhishek.bhr@cs.rutgers.edu
- 2 Tata Institute of Fundamental Research, Mumbai, India  
prahladh@tifr.res.in
- 3 Department of Mathematics, IIT Bombay, Bombay, India  
srikanth@math.iitb.ac.in

---

## Abstract

We study approximation of Boolean functions by low-degree polynomials over the ring  $\mathbb{Z}/2^k\mathbb{Z}$ . More precisely, given a Boolean function  $F : \{0, 1\}^n \rightarrow \{0, 1\}$ , define its  $k$ -lift to be  $F_k : \{0, 1\}^n \rightarrow \{0, 2^{k-1}\}$  by  $F_k(x) = 2^{k-F(x)} \pmod{2^k}$ . We consider the fractional agreement (which we refer to as  $\gamma_{d,k}(F)$ ) of  $F_k$  with degree  $d$  polynomials from  $\mathbb{Z}/2^k\mathbb{Z}[x_1, \dots, x_n]$ .

Our results are the following:

- Increasing  $k$  can help: We observe that as  $k$  increases,  $\gamma_{d,k}(F)$  cannot decrease. We give two kinds of examples where  $\gamma_{d,k}(F)$  actually increases. The first is an infinite family of functions  $F$  such that  $\gamma_{2d,2}(F) - \gamma_{3d-1,1}(F) \geq \Omega(1)$ . The second is an infinite family of functions  $F$  such that  $\gamma_{d,1}(F) \leq \frac{1}{2} + o(1)$  – as small as possible – but  $\gamma_{d,3}(F) \geq \frac{1}{2} + \Omega(1)$ .
- Increasing  $k$  doesn't always help: Adapting a proof of Green [*Comput. Complexity*, 9(1):16–38, 2000], we show that irrespective of the value of  $k$ , the Majority function  $\text{Maj}_n$  satisfies

$$\gamma_{d,k}(\text{Maj}_n) \leq \frac{1}{2} + \frac{O(d)}{\sqrt{n}}.$$

In other words, polynomials over  $\mathbb{Z}/2^k\mathbb{Z}$  for large  $k$  do not approximate the majority function any better than polynomials over  $\mathbb{Z}/2\mathbb{Z}$ .

We observe that the model we study subsumes the model of *non-classical polynomials* in the sense that proving bounds in our model implies bounds on the agreement of non-classical polynomials with Boolean functions. In particular, our results answer questions raised by Bhowmick and Lovett [*In Proc. 30th Computational Complexity Conf., pages 72–87, 2015*] that ask whether non-classical polynomials approximate Boolean functions better than classical polynomials of the same degree.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** Polynomials over rings, Approximation by polynomials, Boolean functions, Non-classical polynomials

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.12

---

\* A full version of the paper is available at <https://arxiv.org/abs/1701.06268>.

† Research supported in part by UGC-ISF grant 1399/4. Part of the work done when the first author was visiting TIFR.



## 1 Introduction

Many lower bound results in circuit complexity are proved by showing that any small sized circuit in a given circuit class can be approximated by a function from a simple computational model (e.g., small depth circuits by low-degree polynomials) and subsequently showing that this is not possible for some suitable “hard” function.

A classic case in point is the work of Razborov [12] which shows lower bounds for  $\text{AC}^0[\oplus]$ , the class of constant depth circuits made up of AND, OR and  $\oplus$  gates. Razborov shows that any small  $\text{AC}^0[\oplus]$  circuit  $C$  can be well approximated by a low-degree multivariate polynomial  $Q(x_1, \dots, x_n) \in \mathbb{F}_2[x_1, \dots, x_n]$  in the sense that

$$\Pr_{x \sim \{0,1\}^n} [Q(x) \neq C(x)] = o(1).$$

The next step in the proof is to show that the hard function, on the other hand, does not have any such approximation. Razborov does this for a suitable symmetric function, Smolensky [13] for the  $\text{MOD}_q$  function (for constant odd  $q$ ), and Szegedy [15] and Smolensky [14] for the Majority function  $\text{Maj}_n$  on  $n$  bits.

Given the importance of the above lower bound, polynomial approximations in other domains and metrics have been intensely investigated and have resulted in interesting combinatorial constructions and error-correcting codes [8, 4], learning algorithms [11, 9] and more recently in the design of algorithms for combinatorial problems [17, 1] as well.

To describe the model of polynomial approximation considered in this paper, we first recall the Razborov [12] model of polynomial approximation. Given a Boolean function  $F : \{0, 1\}^n \rightarrow \{0, 1\}$  and degree  $d \leq n$ , Razborov considers the largest  $\gamma$  such that there is a degree  $d$  polynomial  $Q \in \mathbb{F}_2[x_1, \dots, x_n]$  that has agreement at least  $\gamma$  with  $F$  (i.e.,  $\Pr_x[Q(x) = F(x)] \geq \gamma$ ). Call this  $\gamma_d(F)$ . In this notation, Szegedy [15] and Smolensky’s [14] results for the Majority function can be succinctly stated as

$$\gamma_d(\text{Maj}_n) \leq \frac{1}{2} + \frac{O(d)}{\sqrt{n}}.$$

We consider a generalization of the above model to rings  $\mathbb{Z}/2^k\mathbb{Z}$  in the following simple manner. To begin with, we consider the ring  $\mathbb{Z}/4\mathbb{Z}$ . Given a Boolean function  $F$ , let  $F_2 : \{0, 1\}^n \rightarrow \{0, 2\} \subseteq \mathbb{Z}/4\mathbb{Z}$  be the 2-lift of  $F$  defined as  $F_2(x) := 2^{2-F(x)}$  (i.e.,  $F_2(x) := 0$  if  $F(x) = 0$  and  $F_2(x) := 2$  otherwise). Once again, we can define  $\gamma_{d,2}(F)$  to be the largest  $\gamma$  such that there exists a degree  $d$  polynomial  $Q_2 \in \mathbb{Z}/4\mathbb{Z}[x_1, \dots, x_n]$  that has agreement  $\gamma$  with  $F_2$ . Note that  $\gamma_{d,2}(F) \geq \gamma_d(F)$  since if, for instance,  $Q(x) = x_1x_2 + x_3 \in \mathbb{F}_2[x_1, \dots, x_n]$  has agreement  $\gamma$  with  $F$ , then  $Q_2 := 2(x_1x_2 + x_3) \in \mathbb{Z}/4\mathbb{Z}[x_1, \dots, x_n]$  also has the same agreement  $\gamma$  with  $F_2$ . Hence, proving upper bounds for  $\gamma_{d,2}(F)$  is at least as hard as proving upper bounds for  $\gamma_d(F)$ .

More generally, we can extend these definitions to  $\gamma_{d,k}(F)$ , the agreement of  $F_k$ , the  $k$ -lift of  $F$ , defined as  $F_k(x) = 2^{k-F(x)} \pmod{2^k}$ , with degree  $d$  polynomials from  $\mathbb{Z}/2^k\mathbb{Z}[x_1, \dots, x_n]$ . It is not hard to show that  $\gamma_{d,k+1}(F) \geq \gamma_{d,k}(F)$  and hence as  $k$  increases, the problem of proving upper bounds on  $\gamma_{d,k}(F)$  can only get harder.

Our motivation for this model comes from a recent work of Bhowmick and Lovett [3], who study the maximum agreement between *non-classical polynomials* of degree  $d$  and a Boolean function  $F$ , which is similar to  $\gamma_{d,d}(F)$  (see Section 5 for an exact translation between the above model and non-classical polynomials). In particular, non-classical polynomials of degree  $d$  can be considered as a subset of the degree  $d$  polynomials in  $\mathbb{Z}/2^d\mathbb{Z}[x_1, \dots, x_n]$ .



With respect to correlation<sup>1</sup>, Bhowmick and Lovett showed that there exist non-classical polynomials (and hence polynomials in  $\mathbb{Z}/2^d\mathbb{Z}[x_1, \dots, x_n]$ ) of logarithmic degree that have very good correlation with the  $\text{Maj}_n$  function. With respect to agreement, they show that low-degree non-classical polynomials can only have *small* agreement with the Majority function. Their results stated in our language, imply that

$$\gamma_{d,d}(\text{Maj}_n) \leq \frac{1}{2} + \frac{O(d \cdot 2^d)}{\sqrt{n}}.$$

In particular, if  $d = \Omega(\log n)$ , this result unfortunately does not give any non-trivial bound on the maximum agreement between non-classical polynomials of degree  $d$  and the  $\text{Maj}_n$  function. Bhowmick and Lovett, however, conjectured that this result could be improved and left open the question of whether non-classical polynomials of degree  $d$  can do any better than classical polynomials of the same degree in approximating the Majority function. More generally, they informally conjectured that although non-classical polynomials achieve better correlation with Boolean functions than their classical counterparts, they possibly do not approximate Boolean functions any better than classical polynomials. Our work stems from trying to answer these questions.

## 1.1 Our results

We prove the following results about agreement of Boolean functions with polynomials over the ring  $\mathbb{Z}/2^k\mathbb{Z}$ :

1. We explore whether there exist Boolean functions for which agreement can increase by increasing  $k$ . In particular, do there exist Boolean  $F$  such that  $\gamma_{d,k}(F) > \gamma_{d,1}(F)$ ?

It is not hard to show that this is impossible for  $d = 1$ . Further, it can be shown that if  $\gamma_{d,k}(F) > 1 - \frac{1}{2^d}$ , then  $\gamma_{d,k}(F) = \gamma_{d,1}(F)$ . Keeping this in mind, the first place where we can expect larger  $k$  to show better agreement is  $\gamma_{2,2}$  vs.  $\gamma_{2,1}$ . Our first result shows that there are indeed separating examples in the regime.

- (a) Fix  $d \in \mathbb{N}$  to be any power of 2. For infinitely many  $n$ , there exists a Boolean function  $F : \{0, 1\}^n \rightarrow \{0, 1\}$  such that  $\gamma_{3d-1,1}(F) \leq 5/8 + o(1)$  but  $\gamma_{2d,2}(F) \geq 3/4$ .

Note that since  $F$  is Boolean,  $\gamma_{d,k}(F) \geq 1/2$  for any  $d, k$ . We then ask if there exist Boolean functions  $F$  such that  $\gamma_{d,1}(F)$  is more or less the trivial bound of  $1/2$ , while  $\gamma_{d',k}(F)$  is significantly larger for  $d' \leq d$  and some  $k > 1$ . In this context, we show the following result.

- (b) Fix any  $\ell \geq 2$ . For large enough  $n$ , there is a Boolean function  $F : \{0, 1\}^n \rightarrow \{0, 1\}$  such that  $\gamma_{2^\ell-1,1}(F) \leq 1/2 + o(1)$  but  $\gamma_{d,3}(F) \geq 9/16 - o(1)$ , for  $d = 2^{\ell-1} + 2^{\ell-2} \leq 2^\ell - 1$ .

2. We show that for  $\text{Maj}_n$ , the majority function on  $n$  bits, and any  $d, k \in \mathbb{Z}^+$ ,

$$\gamma_{d,k}(\text{Maj}_n) \leq \frac{1}{2} + \frac{O(d)}{\sqrt{n}},^2$$

by adapting a proof due to Green [7] of a result on the approximability of the parity function by low-degree polynomials over the ring  $\mathbb{Z}/p^k\mathbb{Z}$  for prime  $p \neq 2$ .

<sup>1</sup> The correlation between  $F, G : \{0, 1\}^n \rightarrow \mathbb{Z}/2^k\mathbb{Z}$  is defined to be  $\mathbf{E}_x[\omega^{F(x)-G(x)}]$  where  $\omega$  is the primitive  $2^k$ th root of unity in  $\mathbb{C}$ . If  $F, G$  are  $\{0, 2^{k-1}\}$ -valued, then this quantity is exactly  $2\gamma - 1$  where  $\gamma$  is the agreement between  $F$  and  $G$ . Otherwise, however, it does not measure agreement.

Coupled with the observation that the class of polynomials over rings  $\mathbb{Z}/2^k\mathbb{Z}$  subsumes the class of non-classical polynomials, part (b) of the first result provides a counterexample to an informal conjecture of Bhowmick and Lovett [3] that, for any Boolean function  $F$ , non-classical polynomials of degree  $d$  do not approximate  $F$  any better than classical polynomials of the same degree, and the second result confirms their conjecture that non-classical polynomials do not approximate the Majority function any better than classical polynomials.

## 1.2 Organization

We start with some preliminaries in Section 2. In Section 3, we show some separation results. Next, in Section 4, we prove upper bounds for  $\gamma_{d,k}(\text{Maj}_n)$ . Finally, in Section 5, we discuss how our model relates to non-classical polynomials, answering questions raised by Bhowmick and Lovett.

## 2 Preliminaries

For  $x \in \{0, 1\}^n$ ,  $|x|$  denotes the Hamming weight of  $x$ , and for  $i \geq 0$ ,  $|x|_i$  is the  $(i + 1)$ <sup>th</sup> least significant bit of  $|x|$  in base 2. For  $d \in \mathbb{N}$ , we use  $\{0, 1\}_{\leq d}^n$  (resp.  $\{0, 1\}_{=d}^n$ ) to denote the set of elements in  $\{0, 1\}^n$  of Hamming weight at most  $d$  (resp. exactly  $d$ ). We use  $\mathcal{F}_n$  to denote the collection of all Boolean functions defined on  $\{0, 1\}^n$ .

### 2.1 Elementary symmetric polynomials

Recall that for  $t \geq 1$ , the elementary symmetric polynomial of degree  $t$  over  $\mathbb{F}_2$ ,  $S_t(x_1, \dots, x_n)$ , is defined as  $S_t(x_1, \dots, x_n) = \bigoplus_{1 \leq a_1 < \dots < a_t \leq n} x_{a_1} \dots x_{a_t}$ . Here  $\oplus$  denotes addition modulo two. This may be interpreted as

$$S_t(x_1, \dots, x_n) = \binom{|x|}{t} \bmod 2. \quad (1)$$

A direct consequence of Lucas theorem (see, e.g., [10, Section 1.2.6, Ex. 10]) and Equation (1) is the following:

► **Lemma 2.1.** *For every  $\ell \geq 0$ ,  $S_{2^\ell}(x) = |x|_\ell$ . More generally,  $S_t(x) = \prod_i |x|_i$  where the product runs over all  $i \geq 0$  such that the  $(i + 1)$ <sup>th</sup> least significant bit of the binary expansion of  $t$  is 1.*

The following result follows from the work of Green and Tao [6, Theorem 11.3], who build upon the ideas of Alon and Beigel [2].

► **Theorem 2.2** (Alon-Beigel [2]). *Fix  $\ell \geq 0$ . Then, for every multilinear polynomial  $P \in \mathbb{F}_2[x_1, \dots, x_n]$  of degree at most  $2^\ell - 1$ , we have  $\Pr_{x \sim \{0, 1\}^n} [S_{2^\ell}(x) = P(x)] \leq 1/2 + o(1)$ .*

Theorem 2.2 has a nice corollary:

► **Corollary 2.3.** *For every fixed  $\ell \geq 0$ , the functions  $\{S_{2^i}(x)\}_{0 \leq i \leq \ell}$  are almost balanced and almost uncorrelated, i.e.*

- $\forall 0 \leq i \leq \ell, |\Pr[S_{2^i}(x) = 0] - \Pr[S_{2^i}(x) = 1]| = o(1)$
- $\forall a_0, \dots, a_\ell \in \{0, 1\}, |\Pr[\bigwedge_{0 \leq i \leq \ell} S_{2^i}(x) = a_i] - \frac{1}{2^{\ell+1}}| = o(1)$ .

<sup>2</sup> The constant in the  $O(\cdot)$  is an absolute constant.

Combining Corollary 2.3 with Lemma 2.1, we get another useful fact:

► **Lemma 2.4.** *Let  $x$  be uniformly distributed over  $\{0, 1\}^n$ . Then, for every fixed  $r \geq 1$ , the random variables  $\{|x|_i\}_{0 \leq i \leq r-1}$  are almost uniform and almost  $r$ -wise independent i.e.*

- $\forall 0 \leq i \leq r-1, |\Pr[|x|_i = 0] - \Pr[|x|_i = 1]| = o(1)$ .
- $\forall (a_0, \dots, a_{r-1}) \in \{0, 1\}^r, |\Pr[(|x|_0, \dots, |x|_{r-1}) = (a_0, \dots, a_{r-1})] - \frac{1}{2^r}| = o(1)$ .

## 2.2 Boolean functions and polynomials over $\mathbb{Z}/2^k\mathbb{Z}$

Given an  $F \in \mathcal{F}_n$  and  $k \geq 1$ , we define the  $k$ -lift of  $F$  to be the function  $F_k : \{0, 1\}^n \rightarrow \mathbb{Z}/2^k\mathbb{Z}$  defined as follows. For any  $x \in \{0, 1\}^n$ ,

$$F_k(x) = \begin{cases} 0 & \text{if } F(x) = 0, \\ 2^{k-1} & \text{otherwise.} \end{cases}$$

For  $d \in \mathbb{N}, k \geq 1$ ,  $\mathcal{P}_{d,k}$  will denote the set of multilinear polynomials in  $\mathbb{Z}/2^k\mathbb{Z}[x_1, \dots, x_n]$  of degree at most  $d$ .

For functions  $F, G : D \rightarrow R$  for some finite domain  $D$  and range  $R$ , define the *agreement* of  $F$  and  $G$  – denoted  $\text{agr}(F, G)$  – to be the fraction of inputs where they agree: i.e.,

$$\text{agr}(F, G) = \Pr_{x \sim D} [F(x) = G(x)].$$

We will consider how well multilinear polynomials of a certain degree can approximate Boolean functions in the above sense. More precisely, for any Boolean function  $F \in \mathcal{F}_n$ , we define

$$\gamma_{d,k}(F) = \max_{Q \in \mathcal{P}_{d,k}} \text{agr}(F_k, Q).$$

Following [5], we call a set  $I \subseteq \{0, 1\}^n$  an *interpolating set* for  $\mathcal{P}_{d,k}$  if the only polynomial  $P \in \mathcal{P}_{d,k}$  that vanishes at all points in  $I$  is zero everywhere. Formally, for any  $P \in \mathcal{P}_{d,k}$ ,

$$(\forall x \in I \ P(x) = 0) \Rightarrow (\forall y \in \{0, 1\}^n \ P(y) = 0).$$

We now state a number of standard facts regarding Boolean functions and multilinear polynomials over  $\mathbb{Z}/2^k\mathbb{Z}$ . The proofs are either easy or well-known, and appear in the full version of the paper (see <https://arxiv.org/abs/1701.06268>).

Unless mentioned otherwise, let  $n, d, k$  be any integers satisfying  $n \geq 1, d \geq 0, k \geq 1$ .

► **Lemma 2.5.** *Any polynomial  $Q \in \mathcal{P}_{d,k}$  satisfies the following:*

1. *If  $Q$  is non-zero, then  $\Pr_{x \sim \{0,1\}^n} [Q(x) \neq 0] \geq \frac{1}{2^d}$ .*
2.  *$Q$  is the zero polynomial iff  $Q(x) = 0$  for all  $x \in \{0, 1\}^n$ .*
3. (Möbius Inversion) *Say  $Q(x) = \sum_{|S| \leq d} c_S x_S$ , where  $c_S \in \mathbb{Z}/2^k\mathbb{Z}$  and  $x_S$  denotes  $\prod_{i \in S} x_i$ . Then,  $c_S = \sum_{T \subseteq S} (-1)^{|S|-|T|} Q(1_T)$  where  $1_T \in \{0, 1\}^n$  is the characteristic vector of  $T$ .*
4. *( $\{0, 1\}_{\leq d}^n$  is an interpolating set)  $Q$  vanishes at all points in  $\{0, 1\}^n$  iff  $Q$  vanishes at all points of  $\{0, 1\}_{\leq d}^n$ . By shifting the origin to any point of  $\{0, 1\}^n$ , the same is true of any Hamming ball of radius  $d$  in  $\{0, 1\}^n$ .*

► **Lemma 2.6.** *Fix any  $F \in \mathcal{F}_n$ .*

1.  $\gamma_{d,k}(F) \geq \frac{1}{2}$ .
2.  $\gamma_{d,k+1}(F) \geq \gamma_{d,k}(F)$ .
3.  $\gamma_{d,k}(F) > 1 - \frac{1}{2^d} \Rightarrow \gamma_{d,k}(F) = \gamma_{d,1}(F)$ .
4.  $\gamma_{1,k}(F) = \gamma_{1,1}(F)$ .

### 3 Some separation results

#### 3.1 A separation at $k = 2$

Let  $d \in \mathbb{N}$  be any power of 2. In this section, we show that there are functions  $F$  for which  $\gamma_{2d,2}(F) > \gamma_{3d-1,1}(F)$ .

► **Theorem 3.1.** *For large enough  $n$ , there exists a function  $F \in \mathcal{F}_{2n}$  such that  $\gamma_{2d,2}(F) \geq \frac{3}{4} - o(1)$  but  $\gamma_{3d-1,1}(F) \leq \frac{5}{8} + o(1)$ .*

In particular, we see that  $\gamma_{2,2}(F) > \gamma_{2,1}(F)$ . This result is notable, since it shows that there is a separation at the first place where it is possible to have one (Recall that  $\gamma_{1,k}(F) = \gamma_{1,1}(F)$  for any  $F \in \mathcal{F}_n$  by Lemma 2.6).

Let us begin the proof of Theorem 3.1. We first define a family of Boolean functions on  $\{0,1\}^{2n}$ . We denote the  $2n$  variables by  $x_1, \dots, x_n$  and  $y_1, \dots, y_n$ . We use  $\binom{|x|}{d}$  to denote the  $d$ th elementary symmetric polynomial from the ring  $\mathbb{Z}/4\mathbb{Z}[x_1, \dots, x_n]$ , i.e.,  $\binom{|x|}{d} = \sum_{S \in \binom{[n]}{d}} \prod_{i \in S} x_i$ .<sup>3</sup>

The following theorem due to Kummer (see, e.g., [10, Section 1.2.6, Ex. 11]) determines the largest power of a prime that divides a binomial coefficient.

► **Theorem 3.2 (Kummer).** *Let  $p$  be a prime and  $N, M \in \mathbb{N}$  such that  $N \geq M$ . Suppose  $r$  is the largest integer such that  $p^r \mid \binom{N}{M}$ . Then  $r$  is equal to the number of borrows required when subtracting  $M$  from  $N$  in base  $p$ .*

We will need the following easy corollary of Kummer's theorem.

► **Corollary 3.3.** *Let  $d$  be a power of 2. Then, for  $N \geq d$ , the highest power of 2 dividing  $\binom{N}{d}$  is equal to the highest power of 2 dividing  $\lfloor \frac{N}{d} \rfloor$ .*

Let  $S = \{(x, y) \mid \binom{|x|}{d}, \binom{|y|}{d} \equiv 1 \pmod{2}\}$ . Given any function  $H : \{0,1\}^{2n} \rightarrow \{0,1\}$ , we define the Boolean function  $F_H(x_1, \dots, x_n, y_1, \dots, y_n)$  as follows:

$$F_H(x, y) = \begin{cases} 0 & \text{if } \binom{|x|}{d} \cdot \binom{|y|}{d} \equiv 0 \pmod{4}, \\ 1 & \text{if } \binom{|x|}{d} \cdot \binom{|y|}{d} \equiv 2 \pmod{4}, \\ H(x, y) & \text{otherwise.} \end{cases}$$

Define  $P(x, y) = \binom{|x|}{d} \cdot \binom{|y|}{d} \in \mathbb{Z}/4\mathbb{Z}[x_1, \dots, x_n, y_1, \dots, y_n]$ . Note that  $F_H(x, y)$  is defined so that its 2-lift agrees with  $P(x, y)$  on points  $(x, y)$  where  $P(x, y) \in \{0, 2\}$ . Also Corollary 3.3 implies that the following is an alternate equivalent definition of  $F_H$  in terms of elementary symmetric polynomials modulo 2.

$$F_H(x, y) = \begin{cases} 0 & \text{if } S_d(x) = S_d(y) = 0, \\ S_{2d}(y) & \text{if } S_d(x) = 1 \text{ and } S_d(y) = 0, \\ S_{2d}(x) & \text{if } S_d(x) = 0 \text{ and } S_d(y) = 1, \\ H(x, y) & \text{otherwise.} \end{cases} \quad (2)$$

First of all, let us note that for any choice of  $H$ , we have:

► **Lemma 3.4.**  $\gamma_{2d,2}(F_H) \geq \frac{3}{4} - o(1)$ .

<sup>3</sup> We distinguish between  $\binom{|x|}{d}$  and  $S_d(x)$  since the former is from  $\mathbb{Z}/4\mathbb{Z}[x_1, \dots, x_n]$  and latter a polynomial in  $\mathbb{F}_2[x_1, \dots, x_n]$ .

**Proof.** Consider the polynomial  $P(x, y) \in \mathcal{P}_{2d,2}$  defined above. From Equation (2), it follows that the probability that  $P(x, y) \neq F_{H,2}(x, y)$ <sup>4</sup> is less than or equal to the probability that  $S_d(x) = S_d(y) = 1$ , which is  $\frac{1}{4} + o(1)$  by Corollary 2.3. This gives the claim. ◀

The main lemma is the following.

► **Lemma 3.5.** *Say  $H : \{0, 1\}^{2n} \rightarrow \{0, 1\}$  is chosen uniformly at random. Then,*

$$\Pr_H \left[ \gamma_{3d-1,1}(F_H) > \frac{5}{8} + o(1) \right] = o(1).$$

This will prove Theorem 3.1.

The outline of the proof of the above lemma is as follows. Fix any polynomial  $Q \in \mathbb{F}_2[x_1, \dots, x_n, y_1, \dots, y_n]$  of degree at most  $3d-1$ . We need to show that  $\text{agr}(F_H, Q) \leq \frac{5}{8} + o(1)$ . The function  $H : \{0, 1\}^{2n} \rightarrow \{0, 1\}$  we choose will be a *random* function, which ensures that any  $Q$  cannot agree with  $H$  on significantly more than half the inputs in  $S$ . For inputs outside  $S$ , we need a more involved argument, following Alon and Beigel [2]. We show that for any  $Q$  we can find somewhat large sets  $I$  and  $J$  of  $x$  and  $y$  variables respectively such that when we set the variables outside  $I \cup J$ , we obtain a polynomial that is symmetric in the variables of  $I \cup J$ . This is a Ramsey theoretic argument à la Alon-Beigel [2].

Following this argument, we only need to prove the agreement upper bound for  $Q$  that is symmetric in  $x$  and  $y$  variables. This can be done by reduction to a constant-sized problem. A careful computation to solve the constant-sized problem finishes the proof.

The complete technical details of the proof of Lemma 3.5 appear in the full version of the paper.

### 3.2 Symmetric functions as separating examples

We know from Theorem 2.2 that, for every fixed  $\ell \geq 2$ ,  $\gamma_{2^\ell-1,1}(S_{2^\ell}) \leq \frac{1}{2} + o(1)$ . In contrast, the main result of this section shows that

► **Theorem 3.6.** *For every fixed  $\ell \geq 2$ ,  $\gamma_{d,3}(S_{2^\ell}) \geq \frac{9}{16} - o(1)$ , where  $d = 2^{\ell-1} + 2^{\ell-2}$ .*

Notice that  $2^{\ell-1} + 2^{\ell-2} \leq 2^\ell - 1$  for  $\ell \geq 2$ . This implies that, for  $\ell \geq 2$ ,  $S_{2^\ell}(x)$  is an example of a function  $F$  for which there exist  $k, d \in \mathbb{N}$  such that  $\gamma_{d,1}(F) \leq \frac{1}{2} + o(1)$  but  $\gamma_{d',k}(F) \geq \frac{1}{2} + \Omega(1)$  for some  $d' \leq d$ .

**Proof of Theorem 3.6.** Lemma 2.1 from Section 2 tells us that  $S_{2^\ell}(x) = |x|_\ell$ . Thus,  $S_{2^\ell,3}(x) \in \mathbb{Z}/8\mathbb{Z}[x_1, \dots, x_n]$ , the 3-lift of  $S_{2^\ell}(x)$ , is given by

$$S_{2^\ell,3}(x) = \begin{cases} 4 & \text{if } |x|_\ell = 1 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Fix  $d$  to be  $2^{\ell-1} + 2^{\ell-2}$  and consider the polynomial  $P(x) = \sum_{T \subseteq [n]: |T| \leq d} \prod_{i \in T} x_i$  in  $\mathbb{Z}/8\mathbb{Z}[x_1, \dots, x_n]$ . To prove the theorem, it suffices to show that

$$\Pr_{x \sim \{0,1\}^n} [P(x) = S_{2^\ell,3}(x)] \geq \frac{1}{2} + \frac{1}{16} - o(1).$$

<sup>4</sup>  $F_{H,2}$  denotes the 2-lift of  $F_H$ .

**12:8 On Polynomial Approximations Over  $\mathbb{Z}/2^k\mathbb{Z}$**

Clearly,  $P(x) = \binom{|x|}{d} \bmod 8$ , and

$$P(x) = \begin{cases} 0 & \text{if } 8 \mid \binom{|x|}{d} \\ 4 & \text{if } 4 \mid \binom{|x|}{d} \text{ but } 8 \nmid \binom{|x|}{d} \end{cases} \quad (4)$$

Let  $B(x)$  be the number of borrows required when subtracting  $d$  from  $|x|$ . Rewriting (4) in terms of  $B(x)$  using Kummer's theorem (See Theorem 3.2), we get

$$P(x) = \begin{cases} 4 & \text{if } B(x) = 2 \\ 0 & \text{if } B(x) \geq 3 \end{cases} \quad (5)$$

We will need the following lemma.

► **Lemma 3.7.**  $P(x) = S_{2^\ell, 3}(x)$  if

1.  $|x|_{\ell-2} = 0$ , or
2. if  $(|x|_{\ell-2}, |x|_{\ell-1}, |x|_\ell, |x|_{\ell+1}) = (1, 0, 0, 0)$ .

**Proof.** Since  $d = 2^{\ell-1} + 2^{\ell-2}$ , all the bits of  $d$  except  $d_{\ell-1}$  and  $d_{\ell-2}$  are zero. An important observation is that, when subtracting  $d$  from  $|x|$ , no borrows are required by the bits  $|x|_i$ ,  $0 \leq i \leq \ell - 3$ .

Using the above observation, the reader can verify that when  $(|x|_{\ell-2}, |x|_{\ell-1}, |x|_\ell, |x|_{\ell+1}) = (1, 0, 0, 0)$  the number of borrows required is at least 3 i.e.  $B(x) \geq 3$ , which in turn implies that  $P(x) = 0$ . Since  $|x|_\ell = 0$ ,  $S_{2^\ell, 3}(x) = 0$ . This proves the second part of the lemma.

To prove the first part, suppose  $|x|_{\ell-2} = 0$ . Since  $d_{\ell-1} = d_{\ell-2} = 1$ , it follows that both  $|x|_{\ell-2}$  and  $|x|_{\ell-1}$  will need to borrow when subtracting  $d$  from  $|x|$ . As argued before, no borrows are required by the bits before (i.e. less significant than)  $|x|_{\ell-2}$ , and thus the total number of borrows required by the bits  $|x|_i$ ,  $0 \leq i \leq \ell - 1$ , is 2.

Note that the bit  $|x|_{\ell-1}$  borrows from  $|x|_\ell$ . Consider the following case analysis:

- Case  $|x|_\ell = 1$ :  $|x|_\ell$  will not need to borrow since  $d_\ell = 0$ . In fact, none of the bits after (i.e. more significant than)  $|x|_\ell$  will need to borrow, and thus  $B(x) = 2$ . This implies that  $P(x) = 4$ . We also have  $S_{2^\ell, 3}(x) = 4$  and hence  $P(x) = S_{2^\ell, 3}(x)$ .
- Case  $|x|_\ell = 0$ :  $|x|_\ell$  will require a borrow and this means  $B(x) \geq 3$ . This would imply  $P(x) = 0$ . Since  $|x|_\ell = 0$ , it follows that  $P(x) = S_{2^\ell, 3}(x)$ .

This completes the proof. ◀

By Lemma 3.7, we have

$$\Pr[P(x) = S_{2^\ell, 3}(x)] \geq \Pr[|x|_{\ell-2} = 0] + \Pr[(|x|_{\ell-2}, |x|_{\ell-1}, |x|_\ell, |x|_{\ell+1}) = (1, 0, 0, 0)] \quad (6)$$

Using Lemma 2.4 from Section 2, we have

$$\begin{aligned} \Pr[|x|_{\ell-2} = 0] &\geq \frac{1}{2} - o(1) \\ \Pr[(|x|_{\ell-2}, |x|_{\ell-1}, |x|_\ell, |x|_{\ell+1}) = (1, 0, 0, 0)] &\geq \frac{1}{16} - o(1) \end{aligned}$$

which, together with (6), implies

$$\Pr[P(x) = S_{2^\ell, 3}(x)] \geq \frac{1}{2} + \frac{1}{16} - o(1). \quad \blacktriangleleft$$

**4 Upper bounds for  $\gamma_{d,k}(\text{Maj}_n)$**

In this section, we show an upper bound on  $\gamma_{d,k}(\text{Maj}_n)$  where  $\text{Maj}_n$  denotes the Majority function on  $n$  bits.<sup>5</sup>

► **Theorem 4.1.** *For any  $k \geq 1, d \in \mathbb{Z}^+, \gamma_{d,k}(\text{Maj}_n) \leq \frac{1}{2} + \frac{10d}{\sqrt{n}}$ .*

The proof of Theorem 4.1 presented below is an adaptation of techniques appearing in a work of Green [7], who proved a similar result on the approximability of the parity function by polynomials over the ring  $\mathbb{Z}/p^k\mathbb{Z}$ , for prime  $p \neq 2$ .

We will need some definitions and facts about  $\mathcal{P}_{d,k}$ .

We use  $\pi$  to denote the unique ring homomorphism from  $\mathbb{Z}/2^k\mathbb{Z}$  to  $\mathbb{Z}/2\mathbb{Z}$ . Its kernel  $\pi^{-1}(0) = \{a \in \mathbb{Z}/2^k\mathbb{Z} \mid 2^{k-1}a = 0\}$  is the set of non-invertible elements in  $\mathbb{Z}/2^k\mathbb{Z}$ .

We call a set  $S \subseteq \{0, 1\}^n$  *forcing* for  $\mathcal{P}_{d,k}$  if any polynomial  $P \in \mathcal{P}_{d,k}$  that vanishes over  $S$  is forced to take a value in  $\pi^{-1}(0)$  at all points  $x \in \{0, 1\}^n$ . Formally,

$$(\forall x \in S \ P(x) = 0) \Rightarrow (\forall y \in \{0, 1\}^n \ \pi(P(y)) = 0).$$

Define the polynomial  $\pi(P) \in \mathbb{Z}/2\mathbb{Z}[x_1, \dots, x_n]$  to be the polynomial obtained by applying the map  $\pi$  to each of the coefficients of  $P$ . Since a multilinear polynomial in  $\mathbb{Z}/2^k\mathbb{Z}[x_1, \dots, x_n]$  is the zero polynomial iff it vanishes at all points of  $\{0, 1\}^n$  (by Lemma 2.5), we see that  $S$  is forcing iff  $(\forall x \in S \ P(x) = 0) \Rightarrow \pi(P) = 0$ .

Note that any interpolating set for  $\mathcal{P}_{d,k}$  (see Theorem 2 for the definition) is forcing for  $\mathcal{P}_{d,k}$ , but the converse need not be true.

We now adapt the proof of Lemma 11 in [7] to bound the size of forcing sets for  $\mathcal{P}_{d,k}$ .

► **Lemma 4.2.** *If  $S$  is forcing for  $\mathcal{P}_{d,k}$ , then  $|S| \geq |\{0, 1\}_{\leq d}^n| = \binom{n}{\leq d}$ .*

The proof of the above lemma appears in the full version of the paper.

We now use Lemma 4.2 to prove Theorem 4.1.

**Proof of Theorem 4.1.** We assume throughout that  $1 \leq d \leq \frac{\sqrt{n}}{10}$ ; otherwise, there is nothing to prove. Let  $\text{Maj}_{n,k} : \{0, 1\}^n \rightarrow \mathbb{Z}/2^k\mathbb{Z}$  be the  $k$ -lift of the  $\text{Maj}_n$  function. Let  $P \in \mathcal{P}_{d,k}$  be arbitrary and let  $S_P = \{x \in \{0, 1\}^n \mid P(x) = \text{Maj}_{n,k}(x)\}$ . We want to show that  $|S_P| \leq 2^n \cdot (\frac{1}{2} + \frac{10d}{\sqrt{n}})$ . We will argue by contradiction. So assume that  $|S_P| > 2^n \cdot (\frac{1}{2} + \frac{10d}{\sqrt{n}})$ .

Let  $E_P$  be the complement of  $S_P$ , i.e. the set of points where  $P$  makes an error in computing  $\text{Maj}_{n,k}$ . We have  $|E_P| < 2^n(\frac{1}{2} - \frac{10d}{\sqrt{n}})$ . We will try to find a degree  $D$  (for suitable  $D \leq \lfloor n/2 \rfloor$ ) polynomial  $Q$  such that  $Q$  vanishes at all points in  $E_P$  but has the property that  $Q(x)$  is a unit (i.e.  $\pi(Q(x)) \neq 0$ ) for some  $x \in \{0, 1\}^n$ . To be able to do this, we need the fact that  $E_P$  is not forcing for  $\mathcal{P}_{D,k}$ . By Lemma 4.2, if  $E_P$  is indeed forcing for  $\mathcal{P}_{D,k}$ , then

$$\begin{aligned} |E_P| &\geq \sum_{i=0}^D \binom{n}{i} = \left( \sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n}{i} \right) - \sum_{i=D+1}^{\lfloor n/2 \rfloor} \binom{n}{i} \\ &\geq 2^{n-1} - (\lfloor n/2 \rfloor - D) \cdot \binom{n}{\lfloor n/2 \rfloor} \\ &\geq 2^n \cdot \left( \frac{1}{2} - \frac{2(\lfloor n/2 \rfloor - D)}{\sqrt{n}} \right) = 2^n \cdot \left( \frac{1}{2} - \frac{4d}{\sqrt{n}} \right) \end{aligned}$$

<sup>5</sup> We define the majority function as  $\text{Maj}_n(x) = 1$  iff  $|x| > n/2$ .

## 12:10 On Polynomial Approximations Over $\mathbb{Z}/2^k\mathbb{Z}$

where the last equality follows if we choose  $D = \lfloor n/2 \rfloor - 2d$ . This contradicts our upper bound on the size of  $|E_P|$ . Hence,  $E_P$  cannot be forcing for  $\mathcal{P}_{D,k}$ . In particular, we can find  $Q$  that vanishes on  $E_P$  and furthermore,  $\pi(Q(x)) \neq 0$  for some  $x \in \{0, 1\}^n$ .

We now claim that  $\pi(Q(x_0)) \neq 0$  for some  $x_0$  of Hamming weight  $> n/2$ . To see this, consider the polynomial  $Q_1 = \pi(Q)$ . By construction of  $Q$ , we know that  $Q_1$  is a non-zero polynomial of degree  $D$ . Hence, by Lemma 2.5,  $Q_1$  is non-zero when restricted to the Hamming ball of radius  $D < n/2$  around the all 1s vector. In particular, this implies that there is an input  $x_0$  of Hamming weight  $> n/2$  where  $Q_1(x_0)$  is non-zero and hence  $\pi(Q(x_0)) \neq 0$ , or equivalently  $2^{k-1}Q(x_0) \neq 0$ . Fix this  $x_0$  for the remainder of the proof. Note that  $x_0 \notin E_P$  since  $Q$  vanishes on  $E_P$ .

Now, consider the polynomial  $R(x) = Q(x) \cdot P(x)$ . We first show that  $R(x) = 0$  for all  $x$  of Hamming weight  $\leq n/2$ . Consider any  $x$  of Hamming weight  $\leq n/2$ . If  $x \in E_P$ , then  $R(x) = 0$  since  $Q(x) = 0$ . On the other hand, if  $x \notin E_P$ , then  $P(x) = \text{Maj}_{n,k}(x) = 0$  since  $x$  has Hamming weight  $\leq n/2$ . Thus,  $R$  vanishes at all inputs of Hamming weight  $\leq n/2$ .

Since the degree of  $R$  is at most  $\deg(Q) + \deg(P) = D + d = (\lfloor n/2 \rfloor - 2d) + d \leq \lfloor n/2 \rfloor - d$  and  $R$  vanishes at all inputs of  $\{0, 1\}_{\leq n/2}^n$ , this implies (by Lemma 2.5) that  $R$  must be 0 everywhere. However, at  $x_0$ ,  $R(x_0) = Q(x_0)P(x_0) = Q(x_0)\text{Maj}_{n,k}(x_0) = 2^{k-1}Q(x_0) \neq 0$ . This yields the desired contradiction.  $\blacktriangleleft$

### 5 Connection to non-classical polynomials

Let  $\mathbb{T} = \mathbb{R}/\mathbb{Z}$  denote the one dimensional torus. Observing that the additive structure of  $\mathbb{F}_2$  is isomorphic to the additive subgroup  $\{0, 1/2\} < \mathbb{T}$ , we can think of a Boolean function  $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$  as a function  $F : \mathbb{F}_2^n \rightarrow \{0, 1/2\}$ , and conversely, a map  $F : \mathbb{F}_2^n \rightarrow \{0, 1/2\}$  as a Boolean function.

Tao and Ziegler [16] give a characterization of non-classical polynomials as follows:

► **Definition 5.1** (Tao and Ziegler [16]). A function  $F : \mathbb{F}_2^n \rightarrow \mathbb{T}$  is a non-classical polynomial of degree  $\leq d$  if and only if it has the following form:

$$F(x_1, \dots, x_n) = \alpha + \sum_{0 \leq e_1, \dots, e_n \leq 1, k \geq 1: \sum_i e_i + (k-1) \leq d} \frac{c_{e_1, \dots, e_n, k} x_1^{e_1} \dots x_n^{e_n}}{2^k} \pmod{1}$$

Here  $\alpha \in \mathbb{T}$ , and  $c_{e_1, \dots, e_n, k} \in \{0, 1\}$  are uniquely determined.  $\alpha$  is called the *shift* of  $F$ , and the largest  $k$  such that  $c_{e_1, \dots, e_n, k} \neq 0$  for some  $(e_1, \dots, e_n) \in \{0, 1\}^n$  is called the *depth* of  $F$ .

Since we are interested in the agreement of a non-classical polynomial with Boolean (i.e.  $\{0, 1/2\}$ -valued) functions, we will only consider polynomials with shift  $\alpha = \frac{A}{2^k}$ , where  $k$  is the depth of the polynomial and  $A \in \{0, \dots, 2^k - 1\}$ .

► **Remark.** Classical polynomials are non-classical polynomials with  $\alpha \in \{0, 1/2\}$  and depth = 1. It is easy to see that every classical polynomial corresponds to a Boolean function. It is also not hard to show that every Boolean function can be represented as a classical polynomial.

The following lemma relates our model to non-classical polynomials (the proof is given in the full version of the paper):

► **Lemma 5.2.** *Let  $F$  be a Boolean function, and  $d, k \in \mathbb{Z}^+$ ,  $d \geq k$ .*

1. *If there is a non-classical polynomial  $P$  of degree  $d$  and depth  $k$  satisfying  $\text{agr}(F, P) = \gamma$ , then there is a  $P' \in \mathcal{P}_{d,k}$  satisfying  $\text{agr}(F_k, P') = \gamma$ , where  $F_k$  is the  $k$ -lift of  $F$ .*



2. If there is a  $P \in \mathcal{P}_{d,k}$  satisfying  $\text{agr}(F_k, P) = \gamma$ , then there is a non-classical polynomial  $P'$  of degree  $\leq d + k - 1$  and depth  $k$  satisfying  $\text{agr}(F, P') = \gamma$ .

The first part of Lemma 5.2 implies the following corollary of Theorem 4.1:

► **Corollary 5.3.** *Let  $F : \mathbb{F}_2^n \rightarrow \mathbb{T}$  be a non-classical polynomial of degree  $d$ . Then,*

$$\Pr_{x \sim \mathbb{F}_2^n} [\text{Maj}_n(x) = F(x)] \leq \frac{1}{2} + O\left(\frac{d}{\sqrt{n}}\right).$$

This proves a conjecture of Bhowmick and Lovett [3] that non-classical polynomials of degree  $d$  do not approximate the Majority function any better than classical polynomials of the same degree.

The following is a consequence of Theorem 2.2 and the first part of Lemma 5.2:

► **Corollary 5.4.** *Let  $\ell \geq 2$ . Then, for every classical polynomial  $P : \mathbb{F}_2^n \rightarrow \mathbb{T}$  of degree  $\leq 2^\ell - 1$ ,*

$$\Pr_{x \sim \mathbb{F}_2^n} [P(x) = S_{2^\ell}(x)] \leq \frac{1}{2} + o(1).$$

On the other hand, the second part of Lemma 5.2 and Theorem 3.6 imply

► **Corollary 5.5.** *For every  $\ell \geq 2$ , there is a non-classical polynomial  $F : \mathbb{F}_2^n \rightarrow \mathbb{T}$  of degree  $\leq 2^{\ell-1} + 2^{\ell-2} + 2$  and depth 3 such that*

$$\Pr_{x \sim \mathbb{F}_2^n} [F(x) = S_{2^\ell}(x)] \geq \frac{9}{16} - o(1).$$

Noting that  $2^{\ell-1} + 2^{\ell-2} + 2 < 2^\ell$  for  $\ell \geq 4$ , Corollary 5.4 and Corollary 5.5 imply the following:

► **Theorem 5.6.** *There is a Boolean function  $F : \mathbb{F}_2^n \rightarrow \{0, 1/2\}$  and  $d \geq 1$ , such that for every classical polynomial  $P$  of degree at most  $d$ , we have*

$$\Pr_{x \sim \mathbb{F}_2^n} [F(x) = P(x)] \leq \frac{1}{2} + o(1),$$

but there is a non-classical polynomial  $P'$  of degree  $d' \leq d$  satisfying

$$\Pr_{x \sim \mathbb{F}_2^n} [F(x) = P'(x)] \geq \frac{1}{2} + \Omega(1).$$

This provides a counterexample to an informal conjecture of Bhowmick and Lovett [3] that, for any Boolean function  $F$ , non-classical polynomials of degree  $d$  do not approximate  $F$  any better than classical polynomials of the same degree.

**Acknowledgments.** We would like to thank David Barrington for taking the time to explain Szegedy's result [15] to us, Arkadev Chattopadhyay for referring us to Green's result [7], and Swagato Sanyal for helpful discussions. We are also grateful to the anonymous reviewers for their detailed and helpful comments.

## References

- 1 Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *Proc. 26th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 218–230, 2015. doi:10.1137/1.9781611973730.17.
- 2 Noga Alon and Richard Beigel. Lower bounds for approximations by low degree polynomials over  $\mathbb{Z}_m$ . In *Proc. 16th IEEE Conf. on Computational Complexity*, pages 184–187, 2001. doi:10.1109/CCC.2001.933885.
- 3 Abhishek Bhowmick and Shachar Lovett. Nonclassical polynomials as a barrier to polynomial lower bounds. In *Proc. 30th Computational Complexity Conf.*, pages 72–87, 2015. doi:10.4230/LIPIcs.CCC.2015.72.
- 4 Klim Efremenko. 3-query locally decodable codes of subexponential length. *SIAM J. Comput.*, 41(6):1694–1703, 2012. (Preliminary version in *41st STOC*, 2009). doi:10.1137/090772721.
- 5 Parikshit Gopalan. Query-efficient algorithms for polynomial interpolation over composites. *SIAM J. Comput.*, 38(3):1033–1057, 2008. (Preliminary version in *17th SODA*, 2006). doi:10.1137/060661259.
- 6 Ben Joseph Green and Terence Tao. The distribution of polynomials over finite fields, with applications to the Gowers norms. *Contributions to Discrete Mathematics*, 4(2), 2009. URL: <http://cdm.ucalgary.ca/cdm/index.php/cdm/article/view/133>, arXiv:0711.3191.
- 7 Frederic Green. A complex-number Fourier technique for lower bounds on the mod- $m$  degree. *Comput. Complexity*, 9(1):16–38, 2000. (Preliminary version in *12th STACS*, 1995). doi:10.1007/PL00001599.
- 8 Vince Grolmusz. Superpolynomial size set-systems with restricted intersections mod 6 and explicit Ramsey graphs. *Combinatorica*, 20(1):71–86, 2000. doi:10.1007/s004930070032.
- 9 Adam R. Klivans and Rocco A. Servedio. Learning DNF in time  $2^{O(n^{1/3})}$ . *J. Comput. Syst. Sci.*, 68(2):303–318, 2004. (Preliminary version in *33rd STOC*, 2001). doi:10.1016/j.jcss.2003.07.007.
- 10 Donald Ervin Knuth. *Fundamental Algorithms*, volume I of *The Art of Computer Programming*. Addison-Wesley, 3rd edition, 1997.
- 11 Nathan Linial, Yishay Mansour, and Noam Nisan. Constant depth circuits, Fourier transform, and learnability. *J. ACM*, 40(3):607–620, 1993. (Preliminary version in *30th FOCS*, 1989). doi:10.1145/174130.174138.
- 12 Alexander A. Razborov. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition [Russian]. *Matematicheskie Zametki*, 41(4):598–607, 1987. (English translation in *Mathematical Notes of the Academy of Sciences of the USSR*, 41(4):333–338, 1987). URL: <http://mi.mathnet.ru/eng/mz4883>, doi:10.1007/BF01137685.
- 13 Roman Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proc. 19th ACM Symp. on Theory of Computing (STOC)*, pages 77–82, 1987. doi:10.1145/28395.28404.
- 14 Roman Smolensky. On representations by low-degree polynomials. In *Proc. 34th IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 130–138, 1993. doi:10.1109/SFCS.1993.366874.
- 15 Mario Szegedy. *Algebraic Methods in Lower Bounds for Computational Models with Limited Communication*. PhD thesis, University of Chicago, 1989.
- 16 Terence Tao and Tamar Ziegler. The inverse conjecture for the Gowers norm over finite fields in low characteristic. *Ann. Comb.*, 16(1):121–188, 2012. doi:10.1007/s00026-011-0124-3.
- 17 Ryan Williams. New algorithms and lower bounds for circuits with linear threshold gates. In *Proc. 46th ACM Symp. on Theory of Computing (STOC)*, pages 194–202, 2014. doi:10.1145/2591796.2591858.

# $\exists\mathbb{R}$ -Complete Decision Problems about Symmetric Nash Equilibria in Symmetric Multi-Player Games

Vittorio Bilò<sup>1</sup> and Marios Mavronicolas<sup>2</sup>

- 1 Department of Mathematics and Physics, University of Salento, Lecce, Italy  
vittorio.bilo@unisalento.it
- 2 Department of Computer Science, University of Cyprus, Nicosia, Cyprus  
mavronic@cs.ucy.ac.cy

---

## Abstract

We study the complexity of decision problems about symmetric Nash equilibria for symmetric multi-player games. These decision problems concern the existence of a symmetric Nash equilibrium with certain natural properties. We show that a handful of such decision problems are  $\exists\mathbb{R}$ -complete; that is, they are exactly as hard as deciding the *Existential Theory of the Reals*.

**1998 ACM Subject Classification** F.1.1 Models of Computation, F.1.3 Complexity Measures and Classes, F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** Nash equilibrium, complexity of equilibria,  $\exists\mathbb{R}$ -completeness

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.13

## 1 Introduction

### 1.1 The Setting

Algorithmic Game Theory has come to take into account and accommodate computational issues into the classical setting of Game Theory. Thus, one of its major trends seeks to determine tight complexity bounds for algorithmic problems originating from Game Theory. Along this trend, strong emphasis has been put on algorithmic problems about *Nash equilibria* [19, 20], states where no player could improve her payoff by unilateral switching. This is no surprise given that the Nash equilibrium is the most influential equilibrium concept in Game Theory. In this work, we continue the complexity-theoretic study of Nash equilibria. We focus on *decision problems*, asking whether or not a given game has a Nash equilibrium with some natural property; recently, such decision problems about Nash equilibria, whose complexity-theoretic study dates back to the seminal work of Gilboa and Zemel [17], attracted a lot of flourishing interest and attention – see, e.g., [2, 3, 4, 5, 10, 11, 16, 23].<sup>1</sup> The complexity of deciding the existence of *approximate* Nash equilibria with certain properties has been studied in [1, 6, 13, 18].

A key factor affecting the complexity of decision problems about Nash equilibria is the number of players. Due to the rationality of Nash equilibria for bimatrix games, decision problems about Nash equilibria for bimatrix games are placed in  $\mathcal{NP}$ ; on the other hand, there were early found 3-player games whose Nash equilibria were all irrational [19, 20]; under

---

<sup>1</sup> There is a distinct thread of breakthrough results providing exact characterizations of the complexity of the *search problem* about Nash equilibria for  $r$ -player games with  $r \geq 2$  – see [9, 12, 14, 21]. The characterizations amount to completeness for the complexity classes  $\mathcal{PPAD}$  [21] and  $\mathcal{FLXP}$  [14]. By a recent breakthrough result in [22], there is no PTAS for a Nash equilibrium in bimatrix games, assuming the Exponential Time Hypothesis for  $\mathcal{PPAD}$ .



standard complexity-theoretic assumptions, this dashes the hope that decision problems about Nash equilibria for multi-player games could be placed in  $\mathcal{NP}$ . Such considerations suggest that the *tight* bound for the complexity of decision problems about Nash equilibria for multi-player games must be some complexity class encompassing  $\mathcal{NP}$ . Recent pioneering work by Schaefer and Štefankovič [23] identified  $\exists\mathbb{R}$ , a complexity class associated with the *Existential Theory of the Reals* [24], as such a class; it is known that  $\exists\mathbb{R} \subseteq \mathcal{PSPACE}$  [8].

The first decision problem about Nash equilibria shown  $\exists\mathbb{R}$ -complete for multi-player games was the problem asking, given an  $r$ -player game with  $r \geq 3$  and a rational  $\varrho$ , whether there is a Nash equilibrium with no probability exceeding  $\varrho$  [23, Corollary 3.5]. Using this, four additional decision problems were subsequently shown, via a chain of problem-specific reductions,  $\exists\mathbb{R}$ -complete for multi-player games in [16, Section 3]. The present authors presented in [4], via a *single* unifying reduction, a catalog of such  $\exists\mathbb{R}$ -complete problems for multi-player games, encompassing all the decision problems that had been shown  $\mathcal{NP}$ -complete for (symmetric) bimatrix games in [2, 11, 17]; the catalog encompassed the four  $\exists\mathbb{R}$ -complete decision problems for  $r$ -player games with  $r \geq 3$  from [16].

In this work, we focus on *symmetric* multi-player games; symmetric means that all players are identical and indistinguishable: they have the same strategy sets and there is a common payoff function depending only on the player's chosen strategy and on the number of players choosing each strategy. Symmetric games have been studied extensively; already in 1951 Nash proved that every symmetric game has a *symmetric* Nash equilibrium [20]: one where all players play the same mixed strategy. Very recently decision problems about the existence of a symmetric Nash equilibrium with some additional property for a symmetric game were introduced by Garg *et al.* [16]; they showed that deciding the existence of a symmetric Nash equilibrium where all strategies played with non-zero probability are in a given set (resp., a given set of strategies are played with non-zero probability) is  $\exists\mathbb{R}$ -complete for symmetric multi-player games with a constant number  $r \geq 3$  of players [16, Theorem 23].

### 1.1.1 Contribution, Techniques and Significance

As our main result, we present a catalog with ten  $\exists\mathbb{R}$ -complete decision problems about the existence of a symmetric Nash equilibrium with certain properties for symmetric  $r$ -player games with constant  $r \geq 3$  (Theorem 10). Such decision problems include the ones of deciding the existence of a second Nash equilibrium, or of a Nash equilibrium where the expected payoff to each player is at most (resp., at least) a given number. The properties associated with the decision problems in our catalog come originally from the decision problems about Nash equilibria in bimatrix games, which were shown  $\mathcal{NP}$ -complete in [2, 11, 17]; the same properties are also found in the decision problems about the existence of a Nash equilibrium with certain properties for multi-player games, which were shown  $\exists\mathbb{R}$ -complete in [4, 16].

To show the  $\exists\mathbb{R}$ -completeness results, we present a unifying, polynomial time, many-to-one reduction from the decision problem asking about the existence of a symmetric Nash equilibrium where all strategies played with non-zero probability come from a given set  $T$ ; this is  $\exists\mathbb{R}$ -complete [16, Theorem 23]. The many-to-one reduction amounts to a simple *symmetric game reduction*<sup>2</sup> we design and analyze (Section 4), which maps a pair of symmetric  $r$ -player games  $\tilde{G}$  and  $\hat{G}$ , called the *input game* and the *gadget game*, respectively, to a symmetric  $r$ -player game  $G$  with a larger set of strategies;  $G$  is accompanied with a set of strategies  $T$ .

<sup>2</sup> By *symmetric game reduction* we mean any transformation of some game(s) into a symmetric game that preserves certain properties.

Thus,  $\langle \tilde{G}, T \rangle$  represents an instance of the decision problem above, which the many-to-one reduction reduces from; the symmetric game reduction is tailored to this decision problem.

The symmetric game reduction provides certain correspondences between the symmetric Nash equilibria for  $G$  and those for  $\tilde{G}$  and  $\hat{G}$ , respectively. Going backwards, a symmetric Nash equilibrium for  $G$  subsumes a symmetric Nash equilibrium for either  $\tilde{G}$  or  $\hat{G}$  (Lemma 7). Going forward, a symmetric Nash equilibrium for  $\hat{G}$  always induces a symmetric Nash equilibrium for  $G$  (Lemma 8); but a symmetric Nash equilibrium for  $\tilde{G}$  induces a symmetric Nash equilibrium for  $G$  if and only if all strategies played by it with non-zero probability come from the given set of strategies  $T$  (Lemma 9).

As a tool for the symmetric game reduction, we construct and use the symmetric  $r$ -player gadget game  $\hat{G}[m]$ , for any integer  $m \geq 3$  (Section 3).  $\hat{G}[m]$  generalizes the classical *rock-paper-scissors game* to an arbitrary number of players  $r$  and an arbitrary number of strategies  $m \geq 3$ . By construction,  $\hat{G}[m]$  has the crucial property of zero-sum.

We use the forward and backward correspondences between symmetric Nash equilibria given by the symmetric game reduction to conclude that the constructed symmetric game  $G$  has a symmetric Nash equilibrium with the considered property if and only if the input game  $\tilde{G}$  has a symmetric Nash equilibrium where all strategies played by it with non-zero probability come from the given set  $T$  (Lemmas 11, 12, 13 and 14). These yield the  $\exists\mathbb{R}$ -hardness of deciding the existence of a symmetric Nash equilibrium with the property (Theorem 10). These  $\exists\mathbb{R}$ -completeness results essentially settle the chapter on the complexity of decision problems about symmetric Nash equilibria for symmetric  $r$ -player games, with  $r \geq 3$ .

## 1.2 Related Work and Comparison

The symmetric game reduction in Section 4 follows the structure of the *game reduction* in [4, Section 3]. Specifically, what the two reductions have in common is the idea of transforming a pair of a gadget game  $\hat{G}$  and an input game  $\tilde{G}$  into a game  $G$ . Each of the two reductions is such that the Nash equilibria of the gadget game  $\hat{G}$  are always preserved while only the Nash equilibria of the input game that have certain properties are preserved. The game reduction in [4, Section 3] is tailored to the decision problem asking, given a 3-player game and a rational  $\varrho$ , whether there is a Nash equilibrium with no probability exceeding  $\varrho$ ; on the other hand, the symmetric game reduction is tailored to the decision problem asking, given a symmetric 3-player game with a strategy set  $T$ , whether there is a Nash equilibrium where all strategies played with non-zero probability all come from  $T$ .

Our proof techniques (specifically, the symmetric game reduction) apply directly to  $r$ -player symmetric games with  $r \geq 3$ , thanks to the  $r$ -player symmetric gadget game  $G[m]$ . In contrast, the corresponding techniques in [16, Section 4] deal first with symmetric 3-player games [16, Theorems 20 & 21], obtained by employing the technique of *symmetrization* (cf. [7]) to transform a 3-player game into a symmetric 3-player game; then, they apply separately a rather lengthy and complicated reduction from symmetric 3-player games to symmetric  $r$ -player games with  $r > 3$  [16, Theorem 23].<sup>3</sup> Thus, our direct reduction yields a handful of  $\exists\mathbb{R}$ -completeness results while it is much simpler and more elegant and transparent than the long sequence of reductions in [16, Section 4]. Note also that the symmetric game reduction is the *first* symmetric reduction that works entirely within the realm of decision problems

<sup>3</sup> Note that there is no known trivial reduction of symmetric 3-player games to symmetric  $r$ -player games with  $r > 3$ ; this is unlike the case of the trivial reduction of 3-player games to  $r$ -player games with  $r > 3$ , which allows establishing complexity results for decision problems about Nash equilibria for  $r$ -player games with  $r \geq 3$  by focusing on the case  $r = 3$  (cf. [4] and [16, Section 3]).

about Nash equilibria: the reduction in [11], though yielding a symmetric game, involves SAT; the reductions in [4, 16] are not symmetric – in fact, [16] resorts to *symmetrization* (cf. [7]) in order to extend  $\exists\mathbb{R}$ -hardness from arbitrary to symmetric 3-player games.

## 2 Framework

Our presentation closely follows [4, Section 2].

### 2.1 Games and Nash Equilibria

A *game* is a triple  $G = \langle [r], \{\Sigma_i\}_{i \in [r]}, \{U_i\}_{i \in [r]} \rangle$ , where (i)  $[r] = \{1, \dots, r\}$  is a finite set of *players* with  $r \geq 2$ , and (ii) for each player  $i \in [r]$ ,  $\Sigma_i$  is the set of *strategies* for player  $i$ , and  $U_i$  is the *payoff function*  $U_i : \times_{k \in [r]} \Sigma_k \rightarrow \mathbb{R}$  for player  $i$ ;  $G$  is called an *r-player game*.

For each player  $i \in [r]$ , set  $\Gamma_{-i} := \times_{k \in [r] \setminus \{i\}} \Sigma_k$ ; set  $\Gamma := \times_{k \in [r]} \Sigma_k$ . A *profile* is a tuple  $\mathbf{s} \in \Gamma$  of  $r$  strategies, one per player. The vector  $\mathbf{U}(\mathbf{s}) = \langle U_1(\mathbf{s}), \dots, U_r(\mathbf{s}) \rangle$  is the *payoff vector* for  $\mathbf{s}$ . A *partial profile*  $\mathbf{s}_{-i}$  is a tuple of  $r - 1$  strategies, one for each player other than  $i$ ; so  $\mathbf{s}_{-i} \in \Gamma_{-i}$ . For a profile  $\mathbf{s}$  and a strategy  $t \in \Sigma_i$ , denote as  $\mathbf{s}_{-i} \diamond t$  the profile obtained by substituting the strategy  $t$  for  $s_i$  in  $\mathbf{s}$ . Denote as  $\underline{u}(G) = \min_{\mathbf{s} \in \Sigma, i \in [r]} \{U_i(\mathbf{s})\}$  and  $\bar{u}(G) = \max_{\mathbf{s} \in \Sigma, i \in [r]} \{U_i(\mathbf{s})\}$  the *minimum payoff* and the *maximum payoff* for  $G$ , respectively.

The game  $G$  is *symmetric* if  $\Sigma_1 = \Sigma_2 = \dots = \Sigma_r := \Sigma$ , and for each permutation  $\pi$  of the set of players  $[r]$ , for each player  $i \in [r]$ ,  $U_i(s_1, \dots, s_i, \dots, s_r) = U_{\pi(i)}(s_{\pi(1)}, \dots, s_{\pi(i)}, \dots, s_{\pi(r)})$ ; so, the payoff to a player playing a particular strategy is determined by the multiset of strategies played by the other players, and there is no discrimination among the players.

A *mixed strategy* for player  $i \in [r]$  is a probability distribution  $\sigma_i$  on her strategy set  $\Sigma_i$ : a function  $\sigma_i : \Sigma_i \rightarrow [0, 1]$  such that  $\sum_{s \in \Sigma_i} \sigma_i(s) = 1$ . Denote as  $\Delta_i$  the set of mixed strategies on  $\Sigma_i$ . Denote as  $\text{Supp}(\sigma_i)$  the set of strategies  $s \in \Sigma_i$  with  $\sigma_i(s) > 0$ . A *mixed profile*  $\sigma = \{\sigma_i\}_{i \in [r]}$  is a tuple of mixed strategies, one per player. So, a profile is the degenerate case of a mixed profile where all probabilities are either 0 or 1. Set  $\Delta = \Delta(G) := \times_{k \in [r]} \Delta_k$ ; so,  $\sigma \in \Delta$ . A *partial mixed profile*  $\sigma_{-i}$  is a tuple of  $r - 1$  mixed strategies, one per player other than  $i$ . For a mixed profile  $\sigma$  and a mixed strategy  $\tau_i$  of player  $i \in [r]$ , denote as  $\sigma_{-i} \diamond \tau_i$  the mixed profile obtained when substituting  $\tau_i$  for  $\sigma_i$  in the mixed profile  $\sigma$ .

A mixed profile  $\sigma$  induces a probability measure  $\mathbb{P}_\sigma$  on  $\Gamma$  in the natural way; so, for a profile  $\mathbf{s} \in \Gamma$ ,  $\mathbb{P}_\sigma(\mathbf{s}) = \prod_{k \in [r]} \sigma_k(s_k)$ . Say that the profile  $\mathbf{s}$  (resp., the partial profile  $\mathbf{s}_{-i}$ ) is *supported* in the mixed profile  $\sigma$  (resp., the partial mixed profile  $\sigma_{-i}$ ), and write  $\mathbf{s} \sim \sigma$  (resp.,  $\mathbf{s}_{-i} \sim \sigma_{-i}$ ), if  $\mathbb{P}_\sigma(\mathbf{s}) > 0$  (resp.,  $\mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}) > 0$ ). Under the mixed profile  $\sigma$ , the payoff of each player becomes a random variable. So associated with  $\sigma$  is the *expected payoff* for each player  $i \in [r]$ , denoted as  $U_i(\sigma)$ , which is the expectation according to  $\mathbb{P}_\sigma$  of her payoff; so, clearly,  $U_i(\sigma) = \sum_{\mathbf{s} \in \Gamma} \left( \prod_{k \in [r]} \sigma_k(s_k) \right) \cdot U_i(\mathbf{s})$ .

A *pure Nash equilibrium* is a profile  $\mathbf{s} \in \Gamma$  such that for each player  $i \in [r]$  and for each strategy  $t \in \Sigma_i$ ,  $U_i(\mathbf{s}) \geq U_i(\mathbf{s}_{-i} \diamond t)$ . A *mixed Nash equilibrium*, or *Nash equilibrium* for short, is a mixed profile  $\sigma$  such that for each player  $i \in [r]$  and for each mixed strategy  $\tau_i$ ,  $U_i(\sigma) \geq U_i(\sigma_{-i} \diamond \tau_i)$ . Denote as  $\mathcal{NE}(G)$  the set of Nash equilibria for  $G$ . For each game  $G$ ,  $\mathcal{NE}(G) \neq \emptyset$  [19, 20]. We shall make extensive use of the following characterization of Nash equilibria.

► **Lemma 1.** *The mixed profile  $\sigma$  is a Nash equilibrium if and only if for each player  $i \in [r]$ , (1) for each strategy  $t \in \text{Supp}(\sigma_i)$ ,  $U_i(\sigma) = U_i(\sigma_{-i} \diamond t)$ , and (2) for each strategy  $t \notin \text{Supp}(\sigma_i)$ ,  $U_i(\sigma) \geq U_i(\sigma_{-i} \diamond t)$ .*

For an arbitrary number  $\delta$ , denote as  $G + \delta$  the game obtained from  $G$  by adding  $\delta$  to each possible value of the payoff function. We recall a simple fact:

► **Lemma 2.** Consider the pair of  $r$ -player games  $G$  and  $\widehat{G} := G + \delta$ , for some number  $\delta$ . Then,  $\mathcal{NE}(G) = \mathcal{NE}(\widehat{G})$ . Moreover, for each pair of a player  $i \in [r]$  and a Nash equilibrium  $\sigma \in \mathcal{NE}(\widehat{G})$ ,  $\widehat{U}_i(\sigma) = U_i(\sigma) + \delta$ .

A Nash equilibrium  $\sigma$  is *fully mixed* if for each player  $i \in [r]$ ,  $\text{Supp}(\sigma_i) = \Sigma_i$ . A Nash equilibrium is *symmetric* if all mixed strategies are identical. For a symmetric Nash equilibrium  $\sigma$ , denote as  $\text{Supp}(\sigma)$  the common support of all mixed strategies. Denote as  $\mathcal{SN}\mathcal{E}(G)$  the set of symmetric Nash equilibria for  $G$ . For each symmetric game  $G$ ,  $\mathcal{SN}\mathcal{E}(G) \neq \emptyset$  [19, 20].

## 2.2 Decision Problems about Symmetric Nash Equilibria

Here are the formal statements of the decision problems about symmetric Nash equilibria for symmetric games we shall consider; they are given in the style of Garey and Johnson [15], where I. and Q. stand for INSTANCE and QUESTION, respectively.

### ∃ SECOND SNE

I.: A symmetric game  $G$ .

Q.: Is there a second symmetric Nash equilibrium?

### ∃ SNE WITH LARGE PAYOFFS

I.: A symmetric game  $G$  and a number  $u$ .

Q.: Is there a symmetric Nash equilibrium  $\sigma$  s/t for each player  $i \in [r]$ ,  $U_i(\sigma) \geq u$ ?

### ∃ SNE WITH SMALL PAYOFFS

I.: A symmetric game  $G$  and a number  $u$ .

Q.: Is there a symmetric Nash equilibrium  $\sigma$  s/t for each player  $i \in [r]$ ,  $U_i(\sigma) \leq u$ ?

### ∃ SNE WITH LARGE TOTAL PAYOFF

I.: A symmetric game  $G$  and a number  $u$ .

Q.: Is there a symmetric Nash equilibrium  $\sigma$  s/t  $\sum_{i \in [r]} U_i(\sigma) \geq u$ ?

### ∃ SNE WITH SMALL TOTAL PAYOFF

I.: A symmetric game  $G$  and a number  $u$ .

Q.: Is there a symmetric Nash equilibrium  $\sigma$  s/t  $\sum_{i \in [r]} U_i(\sigma) \leq u$ ?

### ∃ SNE WITH LARGE SUPPORTS

I.: A symmetric game  $G$  and an integer  $k \geq 1$ .

Q.: Is there a symmetric Nash equilibrium  $\sigma$  s/t for each player  $i \in [r]$ ,  $|\text{Supp}(\sigma_i)| \geq k$ ?

### ∃ SNE WITH SMALL SUPPORTS

I.: A symmetric game  $G$  and an integer  $k \geq 1$ .

Q.: Is there a symmetric Nash equilibrium  $\sigma$  s/t for each player  $i \in [r]$ ,  $|\text{Supp}(\sigma_i)| \leq k$ ?

### ∃ SNE WITH RESTRICTING SUPPORTS

I.: A symmetric game  $G$  and a subset of strategies  $T \subseteq \Sigma$ .

Q.: Is there a symmetric Nash equilibrium  $\sigma$  s/t  $T \subseteq \text{Supp}(\sigma)$ ?

### ∃ SNE WITH RESTRICTED SUPPORTS

I.: A symmetric game  $G$  and a subset of strategies  $T \subseteq \Sigma$ .

Q.: Is there a symmetric Nash equilibrium  $\sigma$  s/t  $\text{Supp}(\sigma) \subseteq T$ ?

Given two mixed profiles  $\sigma$  and  $\hat{\sigma}$ , denote as  $\text{Diff}(\sigma, \hat{\sigma}) := \{i \in [r] : \sigma_i \neq \hat{\sigma}_i\}$  the set of players with different mixed strategies in  $\sigma$  and  $\hat{\sigma}$ . A Nash equilibrium  $\sigma$  is *Strongly Pareto-Optimal* if for each mixed profile  $\hat{\sigma}$  where there is player  $i \in [r]$  with  $U_i(\hat{\sigma}) > U_i(\sigma)$  for some player  $i \in [r]$ , there is a player  $j \in \text{Diff}(\sigma, \hat{\sigma})$  such that  $U_j(\hat{\sigma}) \leq U_j(\sigma)$ ; so, there is no other profile where at least one player is strictly better off and every player using a different strategy is strictly better off. We have two additional decision problems.

$\exists \neg$  PARETO-OPTIMAL SNE

I.: A symmetric game  $G$ .

Q.: Is there a symmetric Nash equilibrium which is not Pareto-Optimal?

$\exists \neg$  STRONGLY PARETO-OPTIMAL SNE

I.: A symmetric game  $G$ .

Q.: Is there a symmetric Nash equilibrium which is not Strongly Pareto-Optimal?

### 2.3 The Class $\exists\mathbb{R}$

The *Existential Theory of the Reals*, denoted as ETR, is the set of true sentences of the form  $(\exists x_1, \dots, x_n)(\varphi(x_1, \dots, x_n))$ , where  $\varphi$  is a quantifier-free  $(\vee, \wedge, \neg)$ -boolean formula over the signature  $(0, 1, +, *, <, \leq, =)$ , interpreted over the real numbers.  $\exists\mathbb{R}$  is the complexity class associated with ETR: A decision problem *belongs to*  $\exists\mathbb{R}$  if there is a polynomial-time, many-to-one reduction from it to ETR, and it is  $\exists\mathbb{R}$ -hard if there is a polynomial-time many-to-one reduction from each problem in  $\exists\mathbb{R}$  to it; it is  $\exists\mathbb{R}$ -complete if it belongs to  $\exists\mathbb{R}$  and it is  $\exists\mathbb{R}$ -hard. Since satisfiability of a propositional boolean formula (SAT) is expressible in ETR,  $\mathcal{NP} \subseteq \exists\mathbb{R}$ ; so, ETR is for  $\exists\mathbb{R}$  what SAT is for  $\mathcal{NP}$ , and an  $\exists\mathbb{R}$ -complete problem is in  $\mathcal{NP}$  if and only if ETR is in  $\mathcal{NP}$ . We shall use a result from [16, Theorem 23]:

► **Theorem 3** ([16]). *The problems  $\exists$  SNE WITH RESTRICTING SUPPORTS and  $\exists$  SNE WITH RESTRICTED SUPPORTS are  $\exists\mathbb{R}$ -complete for symmetric  $r$ -player games with constant  $r \geq 3$ .*

### 3 The Symmetric Gadget Game $\widehat{G}[m]$

For any two integers  $r$  and  $m$  such that  $r \geq 3$  and  $m \geq 3$ ,  $\widehat{G}[m]$  is a symmetric  $r$ -player game with  $\Sigma = [m]$ . We assume a cyclic ordering on the strategy set  $[m]$  so that strategy  $m$  precedes strategy 1 (i.e., “ $1 - 1 = m$ ”) and strategy 1 follows strategy  $m$  (i.e., “ $m + 1 = 1$ ”). For each strategy  $s \in [m]$ , say that  $s$  *wins* against strategy  $s - 1$ , *loses* against  $s + 1$  and *ties* against any other strategy  $s' \notin \{s - 1, s + 1\}$ . Say that player  $i$  *wins* (resp., *loses* or *ties*) against player  $j$  if  $i$  chooses a strategy which wins (resp., loses or ties) against the strategy chosen by  $j$ . The payoff functions of  $\widehat{G}[m]$  are defined as follows. Fix a profile  $\mathbf{s}$ .

- If  $\bigcup_{i \in [r]} \{s_i\} = \{j, j + 1\}$  for some strategy  $j \in [m]$ , define  $W(\mathbf{s}) := \{i \in [r] : s_i = j + 1\}$  and  $L(\mathbf{s}) := \{i \in [r] : s_i = j\}$ . Each player  $i \in L(\mathbf{s})$  gets payoff  $-1$ ; each player  $i \in W(\mathbf{s})$  gets payoff  $\frac{|L(\mathbf{s})|}{|W(\mathbf{s})|}$ .
- Otherwise all players get payoff 0.

Note that, by construction,  $\widehat{G}[m]$  is zero-sum. Also, by the payoff functions, a player gets a positive (resp., negative) payoff in a profile  $\mathbf{s}$  only if she is choosing a strategy which wins (resp., loses) in the profile  $\mathbf{s}$ . We show:

► **Lemma 4.** *Fix an odd integer  $m \geq 3$ . Then, every symmetric Nash equilibrium  $\sigma$  for  $\widehat{G}[m]$  is fully mixed and has  $\widehat{U}_i(\sigma) = 0$  for each player  $i \in [r]$ .*



Case	Condition on the profile $\mathbf{s}$	$U_i(\mathbf{s})$
(1)	$A(\mathbf{s}) = [r]$	$\tilde{U}_i(\mathbf{s})$
(2)	$C(\mathbf{s}) = [r]$	$\hat{U}_i(\mathbf{s})$
(3)	$s_i \in A$ and $A(\mathbf{s}) \neq [r]$	$\phi - 1$
(4)	$s_i = j \in B$ and $A^\alpha(\mathbf{s}) = [r] \setminus \{i\}$	$\tilde{U}_i(\mathbf{s}_{-i} \diamond (j - n))$
(5)	$s_i = j \in B$ , $A(\mathbf{s}) = [r] \setminus \{i\}$ and $A^\alpha(\mathbf{s}) \neq [r] \setminus \{i\}$	$\theta$
(6)	$i \in B(\mathbf{s})$ and $A(\mathbf{s}) \neq [r] \setminus \{i\}$	$\phi$
(7)	$i \in C(\mathbf{s})$ and $C(\mathbf{s}) \neq [r]$	$\phi + 1$
(8)	None of the above	$\phi - 1$

■ **Figure 1** The payoff functions for the game  $G$ .

Note that in Lemma 4 we are only interested in determining (in order to use later) the cardinality of the support and the utilities in a symmetric Nash equilibrium. Although it may be possible that the symmetric Nash equilibrium is unique, we did not consider this issue since it is beyond our purposes.

## 4 The Symmetric Game Reduction

The *symmetric game reduction* takes as input a pair of symmetric  $r$ -player games, where  $r \geq 3$  is a fixed constant:

- The *input game*  $\tilde{G}$ , with  $\Sigma(\tilde{G}) = [n]$ , coming together with a set of strategies  $T \subseteq \Sigma(\tilde{G})$  with  $|T| = \alpha$ ;  $\tilde{G}$  and  $T$  form together an instance of  $\exists$  SNE WITH RESTRICTED SUPPORTS. Assume, without loss of generality, that  $T$  consists of the first  $\alpha$  strategies in  $\Sigma(\tilde{G})$ .
- The *gadget game*  $\hat{G}$ , with  $\Sigma(\hat{G}) = [m]$ .

It constructs a symmetric  $r$ -player game  $G = G(\tilde{G}, \hat{G})$  having  $\tilde{G}$  and  $\hat{G}$  as *subgames*.

### 4.1 Construction of the Symmetric Game

Set  $\theta := \bar{u}(\tilde{G}) + 1$  and  $\phi := \min \{ \underline{u}(\tilde{G}), \underline{u}(\hat{G}) \} - 1$ . We construct the game  $G$  as follows:

- $\Sigma(G) = [p]$ , with  $p = n + \alpha + m$ . We partition  $[p]$  into three blocks A, B and C as follows:
  - A is the set of the first  $n$  strategies in  $\Sigma(G)$ , which come from  $\Sigma(\tilde{G})$ .
  - C is the set of the last  $m$  strategies in  $\Sigma(G)$ , which come from  $\Sigma(\hat{G})$ .
  - B is the set of the remaining “middle”  $\alpha$  strategies in  $\Sigma(G)$ , which come from  $T$ .

Denote as  $A^\alpha$  the subset of the first  $\alpha$  strategies in A.

- Fix a profile  $\mathbf{s}$ . For any set of strategies  $X \in \{A, A^\alpha, B, C\}$ , set  $X(\mathbf{s}) := \{i \in [r] \mid s_i \in X\}$ ; so  $X(\mathbf{s})$  is the set of players choosing strategies from  $X$  in the profile  $\mathbf{s}$ . The payoff functions are given in Figure 1. Since  $G$  is symmetric, we only need to specify, for a given profile  $\mathbf{s}$ , the payoff  $U_i(\mathbf{s})$  for any fixed player  $i \in [r]$ .

Clearly,  $G$  is constructed in time polynomial in the sizes of  $\tilde{G}$  and  $\hat{G}$ . Note that by Case (1),  $\tilde{G}$  is a subgame of  $G$ ; by Case (2),  $\hat{G}$  is a subgame of  $G$ . So, the blocks A and C correspond to the games  $\tilde{G}$  and  $\hat{G}$ , respectively.

We shall need some notation. For an  $n$ -dimensional vector  $\mathbf{x} \in \mathbb{R}^n$ , denote as  $\vec{\mathbf{x}} \in \mathbb{R}^p$  the  $p$ -dimensional vector with  $\vec{x}_j = x_j$  for  $j \in [n]$  and  $\vec{x}_j = 0$  for  $n < j \leq p$ . Similarly, for an  $m$ -dimensional vector  $\mathbf{x} \in \mathbb{R}^m$ , denote as  $\overleftarrow{\mathbf{x}} \in \mathbb{R}^p$  the  $p$ -dimensional vector with

$\overleftarrow{x}_j = 0$  for  $i \in [n + \alpha]$  and  $\overleftarrow{x}_j = x_j$  for  $n + \alpha < j \leq p$ . Thus, for a mixed profile  $\sigma \in \Delta(\tilde{\mathbf{G}})$ ,  $(\overrightarrow{\sigma}_1, \overrightarrow{\sigma}_2, \dots, \overrightarrow{\sigma}_r) \in \Delta(\mathbf{G})$ ; for a mixed profile  $\sigma \in \Delta(\widehat{\mathbf{G}})$ ,  $(\overleftarrow{\sigma}_1, \overleftarrow{\sigma}_2, \dots, \overleftarrow{\sigma}_r) \in \Delta(\mathbf{G})$ .

## 4.2 Correspondences Between Nash Equilibria

We establish correspondences between the Nash equilibria for  $\tilde{\mathbf{G}}$  and  $\widehat{\mathbf{G}}$ , and those for  $\mathbf{G}$ .

### 4.2.1 Backward Correspondence: From the Game $\mathbf{G}$ to the Subgames

We first prove that a Nash equilibrium for  $\mathbf{G}$  is induced by a Nash equilibrium for either  $\tilde{\mathbf{G}}$  or  $\widehat{\mathbf{G}}$ . We start with two claims about a Nash equilibrium for  $\mathbf{G}$ . We first prove that if some player is playing some strategy outside  $\mathbf{A}$ , then no other player is playing a strategy in  $\mathbf{A}$ .

► **Lemma 5.** *Fix a Nash equilibrium  $\sigma \in \mathcal{NE}(\mathbf{G})$  for which there is a player  $i' \in [r]$  such that  $\text{Supp}(\sigma_{i'}) \setminus \mathbf{A} \neq \emptyset$ . Then, for every player  $i \neq i'$ ,  $\text{Supp}(\sigma_i) \cap \mathbf{A} = \emptyset$ .*

**Proof.** Assume, by way of contradiction, that there is a player  $i \neq i'$  with  $\text{Supp}(\sigma_i) \cap \mathbf{A} \neq \emptyset$ . Fix an arbitrary strategy  $k \in \text{Supp}(\sigma_i) \cap \mathbf{A}$ . Lemma 1 (Condition (1)) implies that

$$U_i(\sigma) = U_i(\sigma_{-i} \diamond k) = \sum_{\mathbf{s}_{-i} \sim \sigma_{-i}} U_i(\mathbf{s}_{-i} \diamond k) \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}).$$

Lemma 1 (Condition (2)) implies that

$$U_i(\sigma) \geq U_i(\sigma_{-i} \diamond (k + n)) = \sum_{\mathbf{s}_{-i} \sim \sigma_{-i}} U_i(\mathbf{s}_{-i} \diamond (k + n)) \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}). \quad (1)$$

Consider the choices of the players other than  $i$  in the partial profile  $\mathbf{s}_{-i}$  supported in  $\sigma_{-i}$ :

(C.1) All of them choose a strategy from  $\mathbf{A}^\alpha$ : Then,  $\mathbf{s}_{-i} \diamond k$  falls into Case (1) of the payoff functions, with  $U_i(\mathbf{s}_{-i} \diamond k) = \tilde{U}_i(\mathbf{s}_{-i} \diamond k)$ ; on the other hand,  $\mathbf{s}_{-i} \diamond (k + n)$  falls into Case (4), with  $U_i(\mathbf{s}_{-i} \diamond (k + n)) = \tilde{U}_i(\mathbf{s}_{-i} \diamond k)$ . Thus,  $U_i(\mathbf{s}_{-i} \diamond k) = U_i(\mathbf{s}_{-i} \diamond (k + n))$ .

(C.2) All of them choose a strategy from  $\mathbf{A}$  and at least one chooses a strategy from  $\mathbf{A} \setminus \mathbf{A}^\alpha$ .

Then,  $\mathbf{s}_{-i} \diamond k$  falls into Case (1) of the payoff functions with  $U_i(\mathbf{s}_{-i} \diamond k) = \tilde{U}_i(\mathbf{s}_{-i} \diamond k) < \theta$ ; on the other hand,  $\mathbf{s}_{-i} \diamond (k + n)$  falls into Case (5), with  $U_i(\mathbf{s}_{-i} \diamond (k + n)) = \theta$ . Thus,  $U_i(\mathbf{s}_{-i} \diamond k) < U_i(\mathbf{s}_{-i} \diamond (k + n))$ .

(C.3) At least one player chooses a strategy outside  $\mathbf{A}$ . Then,  $\mathbf{s}_{-i} \diamond k$  falls into Case (3) of the payoff functions, with  $U_i(\mathbf{s}_{-i} \diamond k) = \phi - 1$ ; on the other hand,  $\mathbf{s}_{-i} \diamond (k + n)$  falls into Case (6), with  $U_i(\mathbf{s}_{-i} \diamond (k + n)) = \phi$ . Thus,  $U_i(\mathbf{s}_{-i} \diamond k) < U_i(\mathbf{s}_{-i} \diamond (k + n))$ .

The assumption that  $\text{Supp}(\sigma_{i'}) \setminus \mathbf{A} \neq \emptyset$  implies that there is at least one profile supported in  $\sigma_{-i} \diamond (k + n)$  falling into case (C.3) above. Hence, the case analysis implies that

$$\sum_{\mathbf{s}_{-i} \sim \sigma_{-i}} U_i(\mathbf{s}_{-i} \diamond k) \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}) < \sum_{\mathbf{s}_{-i} \sim \sigma_{-i}} U_i(\mathbf{s}_{-i} \diamond (k + n)) \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}),$$

or  $U_i(\sigma) < U_i(\sigma_{-i} \diamond (k + n))$ . A contradiction to (1). ◀

We continue to prove that if some player is playing no strategy from  $\mathbf{A}$ , then the remaining  $r - 1$  players are playing only strategies from  $\mathbf{C}$ .

► **Lemma 6.** *Fix a Nash equilibrium  $\sigma \in \mathcal{NE}(\mathbf{G})$  for which there is a player  $i'$  with  $\text{Supp}(\sigma_{i'}) \cap \mathbf{A} = \emptyset$ . Then, for each player  $i \neq i'$ ,  $\text{Supp}(\sigma_i) \subseteq \mathbf{C}$ .*

**Proof.** Assume, by way of contradiction, that there is a player  $i \neq i'$  with  $\text{Supp}(\sigma_i) \setminus C \neq \emptyset$ . Fix an arbitrary strategy  $k \in \text{Supp}(\sigma_i) \setminus C$  and an arbitrary strategy  $h \in C$ . Since  $k \in \text{Supp}(\sigma_i)$ , Lemma 1 (Condition (1)) implies that

$$U_i(\sigma) = U_i(\sigma_{-i} \diamond k) = \sum_{\mathbf{s}_{-i} \sim \sigma_{-i}} U_i(\mathbf{s}_{-i} \diamond k) \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}).$$

Lemma 1 (Condition (2)) implies that

$$U_i(\sigma) \geq U_i(\sigma_{-i} \diamond h) = \sum_{\mathbf{s}_{-i} \in \sigma_{-i}} U_i(\mathbf{s}_{-i} \diamond h) \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}). \quad (2)$$

Fix now a partial profile  $\mathbf{s}_{-i}$  supported in  $\sigma_{-i}$ . Since  $\text{Supp}(\sigma_{i'}) \cap A = \emptyset$ , it follows that  $\mathbf{s}_{-i} \diamond k$  falls into either Case (3) or Case (6) of the payoff functions, with  $U_i(\mathbf{s}_{-i} \diamond k) \leq \phi$ ; on the other hand,  $\mathbf{s}_{-i} \diamond h$  falls into either Case (2) or Case (7), with  $U_i(\mathbf{s}_{-i} \diamond h) \geq \phi + 1$ . This implies that  $U_i(\sigma_{-i} \diamond h) > U_i(\sigma)$ . A contradiction to (2).  $\blacktriangleleft$

We are now ready to prove:

► **Lemma 7.** *Fix a Nash equilibrium  $\sigma \in \mathcal{NE}(\mathbb{G})$ . Then, there are only two possible cases:*

- $\sigma = (\vec{\tau}_1, \vec{\tau}_2, \dots, \vec{\tau}_r)$  for some Nash equilibrium  $\tau \in \mathcal{NE}(\tilde{\mathbb{G}})$ .
- $\sigma = (\overleftarrow{\tau}_1, \overleftarrow{\tau}_2, \dots, \overleftarrow{\tau}_r)$  for some Nash equilibrium  $\tau \in \mathcal{NE}(\widehat{\mathbb{G}})$ .

**Proof.** By Lemmas 5 and 6, either (i)  $\sigma = (\vec{\tau}_1, \vec{\tau}_2, \dots, \vec{\tau}_r)$  for some mixed profile  $\tau \in \Delta(\tilde{\mathbb{G}})$ , or (ii)  $\sigma = (\overleftarrow{\tau}_1, \overleftarrow{\tau}_2, \dots, \overleftarrow{\tau}_r)$  for some mixed profile  $\tau \in \Delta(\widehat{\mathbb{G}})$ . The conditions that  $\tau \in \mathcal{NE}(\tilde{\mathbb{G}})$  and  $\tau \in \mathcal{NE}(\widehat{\mathbb{G}})$  follow from that each of  $\tilde{\mathbb{G}}$  and  $\widehat{\mathbb{G}}$  is a subgame of  $\mathbb{G}$ .  $\blacktriangleleft$

#### 4.2.2 Forward Correspondence: From the Subgames to the Game $\mathbb{G}$

We now characterize the Nash equilibria for the subgames  $\tilde{\mathbb{G}}$  and  $\widehat{\mathbb{G}}$  that induce corresponding Nash equilibria for  $\mathbb{G}$ . Specifically, these are all the Nash equilibria for  $\widehat{\mathbb{G}}$  (Lemma 8) and every Nash equilibrium for  $\tilde{\mathbb{G}}$  where all players play strategies in the restricted support (Lemma 9), respectively. We first prove:

► **Lemma 8.** *Fix a Nash equilibrium  $\sigma \in \mathcal{NE}(\widehat{\mathbb{G}})$ . Then,  $\langle \overleftarrow{\sigma}_1, \overleftarrow{\sigma}_2, \dots, \overleftarrow{\sigma}_r \rangle \in \mathcal{NE}(\mathbb{G})$ .*

**Proof.** By Case (2) of the payoff functions, for each player  $i \in [r]$ ,  $U_i \langle \overleftarrow{\sigma}_1, \overleftarrow{\sigma}_2, \dots, \overleftarrow{\sigma}_r \rangle = \widehat{U}_i(\sigma) \geq \phi + 1$ . Since  $\sigma \in \mathcal{NE}(\widehat{\mathbb{G}})$ , no player could improve her payoff by switching to a strategy from  $C$ . Switching to a strategy outside  $C$  results in a partial mixed profile supporting only profiles from Cases (3) and (6). So the expected payoff of the switching player is at most  $\phi$ .  $\blacktriangleleft$

We continue to prove:

► **Lemma 9.** *Fix a Nash equilibrium  $\sigma \in \mathcal{NE}(\tilde{\mathbb{G}})$ . Then,  $\langle \vec{\sigma}_1, \vec{\sigma}_2, \dots, \vec{\sigma}_r \rangle \in \mathcal{NE}(\mathbb{G})$  if and only if for each player  $i \in [r]$ ,  $\text{Supp}(\sigma_i) \subseteq [\alpha]$ .*

**Proof.** Assume first that  $\langle \vec{\sigma}_1, \vec{\sigma}_2, \dots, \vec{\sigma}_r \rangle \in \mathcal{NE}(\mathbb{G})$ . Fix an arbitrary player  $i \in [r]$  and a strategy  $k \in \text{Supp}(\sigma_i)$ . Then, by Lemma 1 (Condition (1)),

$$\begin{aligned} U_i(\langle \vec{\sigma}_1, \vec{\sigma}_2, \dots, \vec{\sigma}_r \rangle) &= U_i(\sigma_{-i} \diamond k) \\ &= \sum_{\mathbf{s}_{-i} \in \Gamma_{-i}} U_i(\mathbf{s}_{-i} \diamond k) \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}) \\ &= \sum_{\mathbf{s}_{-i} \in \tilde{\Gamma}_{-i}} U_i(\mathbf{s}_{-i} \diamond k) \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}). \end{aligned}$$

### 13:10 $\exists\mathbb{R}$ -Complete Decision Problems about Symmetric Nash Equilibria

Assume, by way of contradiction, that there is a player  $i' \in [r]$  for which  $\text{Supp}(\sigma_{i'}) \not\subseteq [\alpha]$ . Denote as  $\tilde{\Gamma}_{-i}^\alpha$  the set of partial profiles in which all players in  $[r] \setminus \{i\}$  choose a pure strategy in  $[\alpha]$ . Note that the assumption implies that there is at least one partial profile  $\mathbf{s}_{-i} \in \tilde{\Gamma}_{-i} \setminus \tilde{\Gamma}_{-i}^\alpha$  with  $\mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}) > 0$ . Consider now player  $i$  switching to the strategy  $k + n$ . By Cases (4) and (5) of the payoff functions, we get that

$$\begin{aligned}
 U_i(\sigma_{-i} \diamond (k+n)) &= \sum_{\mathbf{s}_{-i} \in \tilde{\Gamma}_{-i}} U_i(\mathbf{s}_{-i} \diamond (k+n)) \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}) \\
 &= \sum_{\mathbf{s}_{-i} \in \tilde{\Gamma}_{-i}} U_i(\mathbf{s}_{-i} \diamond (k+n)) \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}) \\
 &= \sum_{\mathbf{s}_{-i} \in \tilde{\Gamma}_{-i}^\alpha} \tilde{U}_i(\mathbf{s}_{-i} \diamond (k+n)) \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}) \\
 &\quad + \sum_{\mathbf{s}_{-i} \in \tilde{\Gamma}_{-i} \setminus \tilde{\Gamma}_{-i}^\alpha} \tilde{U}_i(\mathbf{s}_{-i} \diamond (k+n)) \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}) \\
 &= \sum_{\mathbf{s}_{-i} \in \tilde{\Gamma}_{-i}^\alpha} \tilde{U}_i(\mathbf{s}_{-i} \diamond k) \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}) + \sum_{\mathbf{s}_{-i} \in \tilde{\Gamma}_{-i} \setminus \tilde{\Gamma}_{-i}^\alpha} \theta \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}) \\
 &> \sum_{\mathbf{s}_{-i} \in \tilde{\Gamma}_{-i}} \tilde{U}_i(\mathbf{s}_{-i} \diamond k) \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}) \\
 &= U_i(\langle \vec{\sigma}_1, \vec{\sigma}_2, \dots, \vec{\sigma}_r \rangle).
 \end{aligned}$$

a contradiction to the assumption that  $\langle \vec{\sigma}_1, \vec{\sigma}_2, \dots, \vec{\sigma}_r \rangle \in \mathcal{NE}(\mathbb{G})$ .

Assume now that for each player  $i \in [r]$ ,  $\text{Supp}(\sigma_i) \subseteq [\alpha]$ . Fix an arbitrary player  $i \in [r]$ . Clearly, by Case (1) of the payoff functions, player  $i$  cannot improve her payoff by switching to a strategy from A. Also, by Case (7) of the payoff functions, player  $i$  cannot improve her payoff by switching to a strategy from C. So it remains to consider player  $i$  switching to the strategy  $h + n \in \mathbb{B}$ , with  $h \in [\alpha]$ . Since  $h \in [\alpha]$ , Lemma 1 (Condition(2)) implies that

$$U_i(\langle \vec{\sigma}_1, \vec{\sigma}_2, \dots, \vec{\sigma}_r \rangle) \geq \sum_{\mathbf{s}_{-i} \in \tilde{\Gamma}_{-i}} \tilde{U}_i(\mathbf{s}_{-i} \diamond h) \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}).$$

By Case (4) of the payoff functions, we get that

$$\begin{aligned}
 U_i(\sigma_{-i} \diamond (h+n)) &= \sum_{\mathbf{s}_{-i} \in \tilde{\Gamma}_{-i}} U_i(\mathbf{s}_{-i} \diamond (h+n)) \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}) \\
 &= \sum_{\mathbf{s}_{-i} \in \tilde{\Gamma}_{-i}} U_i(\mathbf{s}_{-i} \diamond (h+n)) \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}) \\
 &= \sum_{\mathbf{s}_{-i} \in \tilde{\Gamma}_{-i}} \tilde{U}_i(\mathbf{s}_{-i} \diamond h) \cdot \mathbb{P}_{\sigma_{-i}}(\mathbf{s}_{-i}).
 \end{aligned}$$

It follows that  $\langle \vec{\sigma}_1, \vec{\sigma}_2, \dots, \vec{\sigma}_r \rangle \in \mathcal{NE}(\mathbb{G})$ . The proof is now complete.  $\blacktriangleleft$

## 5 The $\exists\mathbb{R}$ -Complete Decision Problems

We now present the  $\exists\mathbb{R}$ -completeness results. We show:

► **Theorem 10.** *Restricted to  $r$ -player symmetric games with constant  $r \geq 3$ , the following decision problems are  $\exists\mathbb{R}$ -complete:*

<i>Group I</i>	<i>Group II</i>
$\exists$ SECOND SNE	$\exists$ SNE WITH LARGE PAYOFFS
$\exists$ SNE WITH SMALL PAYOFFS	$\exists$ SNE WITH LARGE TOTAL PAYOFF
$\exists$ SNE WITH SMALL TOTAL PAYOFF	$\exists$ SNE WITH SMALL SUPPORTS
$\exists$ SNE WITH LARGE SUPPORTS	
$\exists$ SNE WITH RESTRICTING SUPPORTS	
$\exists \neg$ PARETO-OPTIMAL SNE	
$\exists \neg$ STRONGLY PARETO-OPTIMAL SNE	

Membership of the decision problems (for  $r$ -player games with  $r \geq 3$ ) in  $\exists\mathbb{R}$  is established with standard techniques by employing simple ETR formulas to define their properties (cf. [16]).

**Proof.** By reduction from  $\exists$  SNE WITH RESTRICTED SUPPORT (Theorem 3). Consider an instance  $\tilde{G}$  of  $\exists$  SNE WITH RESTRICTED SUPPORT, called the *input game*. Assume, without loss of generality, that  $\tilde{\Sigma} = [n]$ . We start with an informal outline of the proof. The reduction will be the composition of (i) the construction of a gadget game, and (ii) the symmetric game reduction from Section 4. For each of *Group I* and *Group II*, we shall employ a suitable game  $\hat{G}$ , called the *gadget game*, which may be constructed from the input game  $\tilde{G}$ . Then, we apply the symmetric game reduction from Section 4 with  $\tilde{G}$  and  $\hat{G}$  as the subgames to obtain the game  $G := G \langle \tilde{G}, \hat{G} \rangle$ ;  $G$  is the instance of the decision problem (from the corresponding *Group*) associated with some particular property of symmetric Nash equilibria; to prove that the decision problem is  $\exists\mathbb{R}$ -hard, we need to establish: The game  $\tilde{G}$  has a symmetric Nash equilibrium in which all players play strategies from  $[\alpha]$  if and only if the game  $G$  has a symmetric Nash equilibrium with the property (resp., the set of Nash equilibria for  $G$  has the property, as for the decision problem  $\exists$  SECOND SNE). We now present the formal proof. We treat separately each of *Group I* and *Group II*.

**Group I:** Construct the trivial  $r$ -player symmetric gadget game  $\hat{G}$ , where each player  $i \in [r]$  has a unique strategy giving her payoff  $\bar{u}(\tilde{G}) + 1$ . Clearly,  $\hat{G}$  is constructed in time polynomial in the size of the inbox game  $\tilde{G}$ . Furthermore,  $\hat{G}$  has a unique symmetric Nash equilibrium. Apply now the symmetric game reduction from Section 4 to construct the game  $G$  from the subgames  $\tilde{G}$  and  $\hat{G}$ . Since (i)  $G$  is constructed in time polynomial in the sizes of  $\tilde{G}$  and  $\hat{G}$ , and (ii)  $\hat{G}$  is constructed in time polynomial in the size of  $\tilde{G}$ , it follows that  $G$  is constructed in time polynomial in the size of  $\tilde{G}$ . Lemmas 7, 8 and 9 immediately imply:

► **Lemma 11.** *Assume that the inbox game  $\tilde{G}$  has no Nash equilibrium in the restricted support. Then,  $G$  has a unique symmetric Nash equilibrium  $\sigma$  with the following properties: (i) for each player  $i \in [r]$ ,  $U_i(\sigma) = \bar{u}(\tilde{G}) + 1 = \theta$ ; (ii)  $\text{Supp}(\sigma) = \{n + \alpha + 1\}$ ; (iii)  $\sigma$  is Pareto-Optimal; (iv)  $\sigma$  is Strongly Pareto-Optimal.*

On the other hand, Lemma 9 immediately implies:

► **Lemma 12.** *Assume that the inbox game  $\tilde{G}$  has a Nash equilibrium in the restricted support. Then,  $G$  has a Nash equilibrium  $\tau$  with the following properties: (i) for each player  $i \in [r]$ ,  $U_i(\tau) \leq \bar{u}(\tilde{G}) < \theta$ ; (ii)  $\text{Supp}(\tau) \subset [\alpha]$  with  $|\text{Supp}(\tau)| \geq h$  for some integer  $h \geq 2$ ; (iii)  $\tau$  is not Pareto-Optimal; (iv)  $\tau$  is not Strongly Pareto-Optimal.*

Now, given the  $\exists\mathbb{R}$ -hardness of  $\exists$  SNE WITH RESTRICTED SUPPORTS for games with a constant number of players  $r \geq 3$  (Theorem 3), combining corresponding properties from the two families of properties in Lemmas 11 and 12 immediately yields the  $\exists\mathbb{R}$ -hardness of the following decision problems:

## 13:12 $\exists\mathbb{R}$ -Complete Decision Problems about Symmetric Nash Equilibria

- $\exists$  SECOND SNE.
- $\exists$  SNE WITH SMALL PAYOFFS, by taking  $u$  with  $\bar{u}(\tilde{G}) < u \leq \theta$ .
- $\exists$  SNE WITH SMALL TOTAL PAYOFF, by taking  $u$  with  $r \cdot \bar{u}(\tilde{G}) < u \leq r \cdot \theta$ .
- $\exists$  SNE WITH LARGE SUPPORTS, by taking  $k$  with  $2 \leq k \leq h$ .
- $\exists$  SNE WITH RESTRICTING SUPPORTS, by taking  $T := \{s\}$  for any strategy  $s \in [n]$ .
- $\exists$  NON-PARETO-OPTIMAL SNE.
- $\exists$  NON-STRONGLY PARETO-OPTIMAL SNE.

**Group II:** Fix an odd integer  $m > \alpha$  with size polynomial in the size of  $n$ . Construct the symmetric  $r$ -player gadget game  $\hat{G} := G[m] + \underline{u}(\tilde{G}) - 1$ , where  $G[m]$  is the gadget game from Section 3. Clearly, the game  $\hat{G}$  is constructed in time polynomial in the size of  $\tilde{G}$ . (By Lemmas 2 and 4, each symmetric Nash equilibrium  $\sigma$  for  $\hat{G}$  is fully mixed and has  $\hat{U}_i(\sigma) = \underline{u}(\tilde{G}) - 1$  for each  $i \in [r]$ .) Apply now the symmetric game reduction from Section 4 to construct the game  $G$  from the subgames  $\tilde{G}$  and  $\hat{G}$ . Note that by construction,  $G$  is constructed in time polynomial in the size of  $\tilde{G}$ . Lemmas 7, 8 and 9 immediately imply:

► **Lemma 13.** *Assume that the input game  $\tilde{G}$  has no Nash equilibrium in the restricted support. Then, each symmetric Nash equilibrium  $\sigma$  for  $G$  has the following properties: (i) for each player  $i \in [r]$ ,  $U_i(\sigma) = \underline{u}(\tilde{G}) - 1$ ; (ii)  $|\text{Supp}(\sigma)| = [m]$ .*

On the other hand, Lemma 9 immediately implies:

► **Lemma 14.** *Assume that the input game  $\tilde{G}$  has a Nash equilibrium in the restricted support. Then,  $G$  has a symmetric Nash equilibrium  $\tau$  with the following properties: (i) for each player  $i \in [r]$ ,  $U_i(\tau) \geq \underline{u}(\tilde{G})$ ; (ii)  $|\text{Supp}(\tau)| \leq \alpha$ .*

Now, given the  $\exists\mathbb{R}$ -hardness of  $\exists$  SNE WITH RESTRICTED SUPPORTS with a constant number of players  $r \geq 3$  (Theorem 3), combining corresponding properties from the two families of properties in Lemmas 13 and 14 immediately yields the  $\exists\mathbb{R}$ -hardness of the following decision problems:

- $\exists$  SNE WITH LARGE PAYOFFS, by taking  $u$  with  $\underline{u}(\tilde{G}) - 1 < u \leq \underline{u}(\tilde{G})$ .
- $\exists$  SNE WITH LARGE TOTAL PAYOFF, by taking  $u$  with  $r \cdot (\underline{u}(\tilde{G}) - 1) < u \leq r \cdot \underline{u}(\tilde{G})$ .
- $\exists$  SNE WITH LARGE SUPPORTS, by taking  $k$  with  $\alpha \leq k < m$ . ◀

We recall that  $\exists$  SNE WITH RESTRICTING SUPPORTS was shown  $\exists\mathbb{R}$ -complete for symmetric  $r$ -player games,  $r \geq 3$  also in [16, Theorem 23] (Theorem 3 in the present paper).

## 6 Conclusion

We presented a handful of  $\exists\mathbb{R}$ -complete decision problems about symmetric Nash equilibria for symmetric  $r$ -player games, where  $r \geq 3$  is a fixed constant, which completely settles their precise complexity. Approximate versions of these decision problems are yet to be considered. There so remain decision problems about symmetric Nash equilibria for symmetric *win-lose*  $r$ -player games, with  $r \geq 3$ . It remains very challenging to identify special classes of symmetric multi-player games where such decision problems about symmetric Nash equilibria (or their approximations) become polynomial time solvable.

## References

- 1 P. Austrin, M. Braverman, and E. Chlamtac. Inapproximability of  $\mathcal{NP}$ -complete variants of Nash equilibrium. *Theory of Computing*, 9:117–142, 2013.
- 2 V. Bilò and M. Mavronicolas. The complexity of decision problems about Nash equilibria in win-lose games. In *Proceedings of the 5th International Symposium on Algorithmic Game Theory*, volume 7615 of *LNCS*, pages 37–48, 2012.
- 3 V. Bilò and M. Mavronicolas. Complexity of rational and irrational Nash equilibria. *Theory of Computing Systems*, 54(3):491–527, 2014.
- 4 V. Bilò and M. Mavronicolas. A catalog of  $\exists\mathbb{R}$ -complete decision problems about Nash equilibria in multi-player games. In *Proceedings of the 33rd Symposium on Theoretical Aspects of Computer Science*, volume 47, pages 17:1–17:13, 2016.
- 5 V. Bonifaci, U. Di Orio, and L. Laura. The complexity of uniform Nash equilibria and related subgraph problems. *Theoretical Computer Science*, 401(1–3):144–152, 2008.
- 6 M. Braverman, Y. Kun-Ko, and O. Weinstein. Approximating the best Nash equilibrium in  $O(n^{o(\lg n)})$  time breaks the exponential time hypothesis. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 970–982, 2015.
- 7 G.W. Brown and J. von Neumann. Solutions of games by differential equations. *Contributions to the Theory of Games, Annals of Mathematics Studies*, (24):73–79, 1950.
- 8 J. Canny. Some algebraic and geometric computations in  $\mathcal{PSPACE}$ . In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 460–467, 1988.
- 9 X. Chen, X. Deng, and S.-H. Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 56(3), 2009.
- 10 B. Codenotti and D. Štefankovič. On the computational complexity of Nash equilibria for  $(0,1)$  bimatrix games. *Information Processing Letters*, 94(3):145–150, 2005.
- 11 V. Conitzer and T. Sandholm. New complexity results about Nash equilibria. *Games and Economic Behavior*, 63(2):621–641, 2008.
- 12 C. Daskalakis, P.W. Goldberg, and C.H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009.
- 13 A. Deligkas, J. Fearnley, and R. Savani. Inapproximability results of approximate Nash equilibria. In *Proceedings of the 12th International Conference on Web and Internet Economics*, volume 10123 of *LNCS*, pages 29–43, 2012.
- 14 K. Etessami and M. Yannakakis. On the complexity of Nash equilibria and other fixed points. *SIAM Journal on Computing*, 39(6):2531–2597, 2010.
- 15 M.J. Garey and D.J. Johnson. *Computers and Intractability – A Guide to the Theory of  $\mathcal{NP}$ -Completeness*. W. H. Freeman, 1979.
- 16 J. Garg, R. Mehta, V.V. Vazirani, and S. Yazdanbod.  $\mathcal{ETR}$ -completeness for decision versions of multi-player (symmetric) Nash equilibria. In *Proceedings of the 42nd International Colloquium on Automata, Languages and Programming*, volume 9134 of *LNCS*, pages 554–566, 2015.
- 17 I. Gilboa and E. Zemel. Nash and correlated equilibria: Some complexity considerations. *Games and Economic Behavior*, 1(1):80–93, 1989.
- 18 E. Hazan and R. Krauthgamer. How hard is it to approximate the best Nash equilibrium? *SIAM Journal on Computing*, 40(1):79–91, 2011.
- 19 J.F. Nash. Equilibrium points in  $n$ -person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36:48–49, 1950.
- 20 J.F. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951.
- 21 C.H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994.

**13:14**  $\exists\mathbb{R}$ -Complete Decision Problems about Symmetric Nash Equilibria

- 22 A. Rubinfeld. Settling the complexity of computing approximate two-player Nash equilibria. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science*, pages 258–265, 2016.
- 23 M. Schaefer and D. Štefankovič. Fixed points, Nash equilibria and the existential theory of the reals. *Theory of Computing Systems*, First online: 04 November 2015.
- 24 A. Tarski. A decision method for elementary algebra and geometry. *RAND Corporation*, 1948.



# On Büchi One-Counter Automata

Stanislav Böhm<sup>\*1</sup>, Stefan Göller<sup>†2</sup>, Simon Halfon<sup>3</sup>, and  
Piotr Hofman<sup>‡4</sup>

1 Technical University of Ostrava, Ostrava, Czech Republic

stanislav.bohm@vsb.cz

2 LSV, CNRS & ENS Cachan, Université Paris-Saclay, Cachan, France

goeller@lsv.fr

3 LSV, CNRS & ENS Cachan, Université Paris-Saclay, Cachan, France

halfon@lsv.fr

4 LSV, CNRS & ENS Cachan, Université Paris-Saclay, Cachan, France

hofman@lsv.fr

---

## Abstract

Equivalence of deterministic pushdown automata is a famous problem in theoretical computer science whose decidability has been shown by Sénizergues. Our first result shows that decidability no longer holds when moving from finite words to infinite words. This solves an open problem that has recently been raised by Löding. In fact, we show that already the equivalence problem for deterministic Büchi one-counter automata is undecidable. Hence, the decidability border is rather tight when taking into account a recent result by Löding and Repke that equivalence of deterministic weak parity pushdown automata (a subclass of deterministic Büchi pushdown automata) is decidable.

Another known result on finite words is that the universality problem for vector addition systems is decidable. We show undecidability when moving to infinite words. In fact, we prove that already the universality problem for nondeterministic Büchi one-counter nets (or equivalently vector addition systems with one unbounded dimension) is undecidable.

**1998 ACM Subject Classification** F.1.1 Models of Computation, F.4.3 Formal Languages

**Keywords and phrases** infinite words, deterministic pushdown automata

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.14

## 1 Introduction

One of the most prominent results in theoretical computer science is the decidability of the equivalence problem for deterministic pushdown automata, shown by Sénizergues [17]. Stirling proved the first complexity bound for this problem [18], namely a tower of exponentials of elementary height, see also [8] for a more recent proof by Jančar. Still there remains a remarkable complexity gap for this problem, to the best of the authors' knowledge the best-known lower bound is P-hardness, which trivially follows from the emptiness problem.

Although a doubly-exponential upper bound has been proved during the nineteen seventies by Valiant [19], too, the regularity problem for deterministic pushdown automata seems to be far from being understood: to the best of the authors' knowledge the best-known lower bound is P-hardness, again trivially following from the emptiness problem.

---

\* Supported by grant GAČR 15-13784S.

† Supported by Labex Digicosme, Univ. Paris-Saclay, project VERICONISS.

‡ Supported by Labex Digicosme, Univ. Paris-Saclay, project VERICONISS.



Although it is unclear whether to lower the upper bound or to raise the lower bound for these central problems, it seems fair to say that we lack techniques to show lower bounds for them. Indeed, the presence of *determinism* seems to be too restrictive to encode computations of Turing machines into instances of the respective problems. To date, we do not even know if equivalence of deterministic pushdown automata is hard for NP – the upper bound of a tower of exponentials of elementary height leaves a huge complexity gap for this problem.

Therefore, subclasses have been studied in the literature for which tight complexity bounds have been obtained, such as deterministic 1-counter [2], 1-counter nets [7], visibly pushdown automata [1], normed context-free processes [3, 4, 6, 5], and height-deterministic pushdown automata [12], to mention a few.

One should bear in mind that for deterministic models universality is typically equally hard as emptiness. For nondeterministic models however universality becomes undecidable very quickly, for instance already for 1-counter automata. Decidability of universality can be regained for such a nondeterministic model by disallowing the automata to test for zero: even universality of vector addition systems can be shown decidable by applying standard well-quasi order arguments. For its subclass 1-counter nets (i.e. 1-counter automata that cannot test for zero) universality has recently been proven to be Ackermann-complete by Hofman and Totzke [7].

**The situation on infinite words.** Unfortunately, equivalence and universality have not gained much attention on infinite words beyond  $\omega$ -regular languages so far, in particular for pushdown automata. Recently Löding and Repke have studied the equivalence problem of deterministic pushdown automata on infinite words and proved that this problem is decidable for weak parity conditions [10, 9, 15] (this is a strict subclass of deterministic Büchi pushdown automata). In fact they showed that given two deterministic weak parity pushdown automata  $\mathcal{A}$  and  $\mathcal{B}$  one can effectively construct two deterministic pushdown automata  $\mathcal{A}'$  and  $\mathcal{B}'$  running on finite words such that  $\mathcal{A}$  and  $\mathcal{B}$  accept the same infinite words if, and only if,  $\mathcal{A}'$  and  $\mathcal{B}'$  accept the same finite words. Extending this decidability result to deterministic parity pushdown automata, or first to the class of deterministic Büchi pushdown automata did not seem to be achievable that easily. In a recent article [9] Löding explicitly raised the question if equivalence of deterministic parity pushdown automata is decidable.

**Our contributions.** In this paper we show that two central decision problems that are decidable on finite words, become undecidable on infinite words. First, we prove that the equivalence problem is already undecidable for deterministic 1-counter automata with a Büchi acceptance condition (even without  $\varepsilon$ -transitions). This solves the above-mentioned question raised by Löding in [9]. This undecidability result can be considered as rather tight since, as mentioned above, equivalence of deterministic weak parity pushdown automata is decidable [10].

Our proof heavily exploits the infinity of the input words and we are confident that our proof technique may be applied to related problems on infinite words. As mentioned before, significant lower bounds involving deterministic automata are typically difficult to prove. In fact, to the best of our knowledge, there are no handy lower bound techniques known for equivalence problem of *deterministic* automata.

Moreover, we consider this undecidability result as somewhat surprising since from a given deterministic 2-counter automaton we construct a deterministic 1-counter automaton such that for all two of its configurations all  $\omega$ -words that distinguish the two configurations must end in the quite restrictive form  $(w\#)^\omega$ , where  $w$  is the unique halting computation of

the initial deterministic 2-counter automaton, if it exists. We contrast our undecidability result on infinite words with a recent NL-completeness result for deterministic 1-counter automata on finite words [2].

As a second result we show that on infinite words universality becomes undecidable on Büchi 1-counter nets. This is shown by a sequence of reductions starting from the boundedness problem for decremental-error vector addition systems, which is undecidable due to Schnoebelen [16]. In the final step of these reductions we again heavily rely on the fact that we employ a Büchi condition.

To the best of the authors' knowledge comparable jumps from decidability on finite words to undecidability on infinite words appear for satisfiability of metric temporal logic [13, 14].

**Organization of this paper.** In Section 2 we introduce the necessary definitions and state our main results. The equivalence problem for deterministic Büchi 1-counter automata is shown to be undecidable in Section 3. Universality of Büchi 1-counter nets is shown to be undecidable in Section 4. We conclude in Section 5. Some of the proofs appear in the appendix due to space restrictions.

## 2 Preliminaries

By  $\mathbb{N} = \{0, 1, \dots\}$  we denote the non-negative integers. For each non-negative integer  $k$  we denote by  $[1, k]$  the set  $\{1, \dots, k\}$ . If  $X$  is a non-empty set and  $\mathbf{x} = (x_1, \dots, x_k) \in X^k$  we write  $\mathbf{x}(i)$  to denote  $x_i$ , i.e. the  $i$ -th component of  $\mathbf{x}$ . We use bold-face font only to denote such vectors. For every integer  $z \in \mathbb{Z}$  we denote its *signum* by  $\text{sgn}(z) = -1$  if  $z < 0$ ,  $\text{sgn}(z) = 0$  if  $z = 0$  and  $\text{sgn}(z) = 1$  if  $z > 0$ . Given a set  $\Sigma$  and some finite word  $w = a_1 \cdots a_n \in \Sigma^n$  with  $a_i \in \Sigma$  we define the infix  $w[i, j] = a_i a_{i+1} \cdots a_j$  and  $w[i] = w[i, i] = a_i$ . Similar remarks apply to infinite words  $w = a_1 a_2 \cdots$ ; moreover we write  $w[i, \infty]$  to denote  $a_i a_{i+1} a_{i+2} \cdots$ .

For each  $k \geq 1$  a  $k$ -counter automaton is a tuple  $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma_{\mathcal{A}}, \delta_{\mathcal{A}}, q_0, F_{\mathcal{A}})$ , where  $Q_{\mathcal{A}}$  is a finite set of *states*,  $\Sigma_{\mathcal{A}}$  is a non-empty finite *alphabet*,  $\delta_{\mathcal{A}} \subseteq Q_{\mathcal{A}} \times \Sigma_{\mathcal{A}} \times \{0, 1\}^k \times \{-1, 0, 1\}^k \times Q_{\mathcal{A}}$  is a set of *transitions*,  $q_0 \in Q_{\mathcal{A}}$  is an *initial state*, and  $F_{\mathcal{A}} \subseteq \delta_{\mathcal{A}}$  is a set of *final transitions*. It is worth mentioning that typically the accepting condition is given by a set of states rather than by a set of transitions. But it is not hard to see that the two formalisms are effectively equivalent: indeed one can translate one formalism to the other in polynomial time.

When it is not of importance we sometimes also drop the last component of  $\mathcal{A}$ . We say  $\mathcal{A}$  is *deterministic* if for every  $(q, a, \boldsymbol{\sigma}) \in Q_{\mathcal{A}} \times \Sigma_{\mathcal{A}} \times \{0, 1\}^k$  there is at most one transition  $\tau = (p, b, \boldsymbol{\mu}, \mathbf{u}, p') \in \delta_{\mathcal{A}}$  with  $p = q$ ,  $b = a$  and  $\boldsymbol{\mu} = \boldsymbol{\sigma}$ . A *configuration* of  $\mathcal{A}$  is an element  $(q, \mathbf{n}) \in Q_{\mathcal{A}} \times \mathbb{N}^k$  that we also write as  $q(\mathbf{n})$  in the following. The bit vector  $\boldsymbol{\sigma} \in \{0, 1\}^k$  that appears in a transition determines which sign each counter is expected to have for the transition to be applicable. More formally, for two configurations  $p(\mathbf{m})$  and  $q(\mathbf{n})$  and a transition of the form  $\tau = (p, a, \boldsymbol{\sigma}, \mathbf{u}, q)$  we write  $p(\mathbf{m}) \xrightarrow{\tau} q(\mathbf{n})$  if  $\boldsymbol{\sigma}(i) = \text{sgn}(\mathbf{m}(i))$  and  $\mathbf{n}(i) = \mathbf{m}(i) + \mathbf{u}(i)$  for all  $i \in [1, k]$ . The relation  $\xrightarrow{\tau}$  is extended to finite words over  $\delta_{\mathcal{A}}$  inductively as follows, where  $\rho \in \delta_{\mathcal{A}}^*$  and  $\tau \in \delta_{\mathcal{A}}$ :  $p(\mathbf{m}) \xrightarrow{\rho} p(\mathbf{m})$  for all configurations  $p(\mathbf{m})$  and  $p(\mathbf{m}) \xrightarrow{\rho\tau} q(\mathbf{n})$  if  $p(\mathbf{m}) \xrightarrow{\rho} r(\boldsymbol{\ell})$  and  $r(\boldsymbol{\ell}) \xrightarrow{\tau} q(\mathbf{n})$  for some configuration  $r(\boldsymbol{\ell})$ . We write  $p(\mathbf{m}) \xrightarrow{*} q(\mathbf{n})$  if  $p(\mathbf{m}) \xrightarrow{\rho} q(\mathbf{n})$  for some  $\rho \in \delta_{\mathcal{A}}^*$ . The following decision problem is well-known to be undecidable in its full generality by [11].

### REACHABILITY

**Input:** A  $k$ -counter automaton  $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma_{\mathcal{A}}, \delta_{\mathcal{A}}, q_0)$  and a control state  $q_f \in Q_{\mathcal{A}}$ .

**Question:**  $q_0(\mathbf{0}) \xrightarrow{*} q_f(\mathbf{0})$ ?

## 14:4 On Büchi One-Counter Automata

For every transition  $\tau = (p, a, \sigma, \mathbf{u}, q)$  let  $\text{READ}(\tau) = a$  denote the letter of the transition. We extend  $\text{READ}$  to a (letter-to-letter) morphism from  $\delta_{\mathcal{A}}^*$  to  $\Sigma_{\mathcal{A}}^*$  in the usual way.

For words  $w \in \Sigma_{\mathcal{A}}^*$  we write  $p(\mathbf{m}) \xrightarrow{w} q(\mathbf{u})$  if  $p(\mathbf{m}) \xrightarrow{\rho} q(\mathbf{n})$  for some  $\rho \in \delta_{\mathcal{A}}^*$  with  $\text{READ}(\rho) = w$ . In this case we also say that  $\rho$  is a *run* of  $\mathcal{A}$  for the word  $w$  from  $p(\mathbf{m})$  to  $q(\mathbf{n})$ . We say that a finite run  $\rho$  is *accepting* if  $\rho \in \delta_{\mathcal{A}}^* F_{\mathcal{A}}$ , i.e. the last transition of  $\rho$  is accepting, otherwise we say  $\rho$  is *rejecting*.

The only acceptance condition on infinite words that we study in this paper is the Büchi acceptance condition. Given an infinite word  $w \in \Sigma_{\mathcal{A}}^\omega$  an  $\omega$ -run of  $\mathcal{A}$  for  $w$  from configuration  $p(\mathbf{m})$  is an infinite sequence  $\rho \in \delta_{\mathcal{A}}^\omega$  such that every finite prefix  $\pi$  of  $\rho$  is a run from  $p(\mathbf{m})$  to some configuration, and  $\text{READ}(\pi)$  is a prefix of  $w$ . We say that the  $\omega$ -run  $\rho$  is *accepting* if  $\rho[i] \in F_{\mathcal{A}}$  for infinitely many  $i \geq 1$ , otherwise we say  $\rho$  is *rejecting*. The *language of a configuration  $p(\mathbf{m})$  of  $\mathcal{A}$*  is defined as

$$L(\mathcal{A}, p(\mathbf{m})) = \{w \in \Sigma_{\mathcal{A}}^* \mid \text{there is an accepting run from } p(\mathbf{m}) \text{ for } w\}.$$

Similarly the  $\omega$ -*language of a configuration  $p(\mathbf{m})$  of  $\mathcal{A}$*  is defined as

$$L_\omega(\mathcal{A}, p(\mathbf{m})) = \{w \in \Sigma_{\mathcal{A}}^\omega \mid \text{there is an accepting } \omega\text{-run from } p(\mathbf{m}) \text{ for } w\}.$$

We sometimes just write  $L(p(\mathbf{m}))$  or  $L_\omega(p(\mathbf{m}))$  if  $\mathcal{A}$  is clear from the context.

Given a finite alphabet  $\Sigma_{\mathcal{A}}$  and a language  $L \subseteq \Sigma_{\mathcal{A}}^*$  (resp.  $\omega$ -language  $R \subseteq \Sigma_{\mathcal{A}}^\omega$ ) we say  $L$  (resp.  $R$ ) is *universal* if  $L = \Sigma_{\mathcal{A}}^*$  (resp.  $R = \Sigma_{\mathcal{A}}^\omega$ ).

We say  $k$ -counter automaton  $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma_{\mathcal{A}}, \delta_{\mathcal{A}}, q_0, F_{\mathcal{A}})$  is a *k-counter net* if the test for zero is not allowed, formally for any transition  $(p, a, \sigma, \mathbf{u}, q) \in \delta_{\mathcal{A}}$  and for any  $\boldsymbol{\mu} \in \{0, 1\}^k$  the transition  $(p, a, \boldsymbol{\mu}, \mathbf{u}, q)$  is in  $\delta_{\mathcal{A}}$  as well. As the third component of any transition of a  $k$ -counter net no longer plays a role we omit it, i.e. we stipulate  $\delta_{\mathcal{A}} \subseteq Q_{\mathcal{A}} \times \Sigma_{\mathcal{A}} \times \{-1, 0, 1\}^k \times Q_{\mathcal{A}}$ . The following monotonicity property holds immediately by definition of  $k$ -counter nets (see for instance [7]).

► **Lemma 1.**  $L(\mathcal{B}, p(\mathbf{m})) \subseteq L(\mathcal{B}, p(\mathbf{n}))$  if  $\mathbf{m} \leq \mathbf{n}$  (and where  $\leq$  is meant component-wise) and  $\mathcal{B}$  is a one-counter net.

The following problems will be of central interest in this paper, where the latter is a special case of the former.

### $\omega$ -LANGUAGE EQUIVALENCE

**Input:** A  $k$ -counter automaton  $\mathcal{A}$  and two configurations  $p(\mathbf{m})$  and  $q(\mathbf{n})$ .

**Question:**  $L_\omega(p(\mathbf{m})) = L_\omega(q(\mathbf{n}))$ ?

Our first main result is the following.

► **Theorem 2.**  $\omega$ -LANGUAGE EQUIVALENCE is undecidable for deterministic 1-counter automata.

### $\omega$ -UNIVERSALITY

**Input:** A  $k$ -counter automaton  $\mathcal{A}$  and a configuration  $p(\mathbf{m})$ .

**Question:**  $L_\omega(p(\mathbf{m})) = \Sigma_{\mathcal{A}}^\omega$ ?

Our second main result is the following.

► **Theorem 3.**  $\omega$ -LANGUAGE UNIVERSALITY is undecidable for 1-counter nets.

### 3 The equivalence problem for deterministic Büchi 1-counter automata is undecidable

We reduce from REACHABILITY for deterministic 2-counter automata, which is undecidable [11]. Our construction does not rely on the fact that our 2-counter automaton is deterministic but it makes the constructions easier to state as distinguishing words of two configurations of the constructed deterministic Büchi 1-counter automaton must repeatedly encounter the unique sequence of transitions from the initial configuration  $q_0(0,0)$  to the configuration  $q_f(0,0)$ .

Let us therefore fix some deterministic 2-counter automaton  $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma_{\mathcal{A}}, \delta_{\mathcal{A}}, q_0)$  and some state  $q_f \in Q_{\mathcal{A}}$ . We show how to construct a deterministic 1-counter automaton  $\mathcal{B} = (Q, \Sigma, \delta, q_0, F)$ , where  $Q$  will contain state  $q_0^{(1)}$  and  $q_0^{(2)}$  such that

$$q_0(0,0) \xrightarrow{*} q_f(0,0) \quad \text{if, and only if,} \quad L_{\omega}(q_0^{(1)}(0)) \neq L_{\omega}(q_0^{(2)}(0)) .$$

We can assume without loss of generality that  $\Sigma_{\mathcal{A}} = \{a\}$  is a singleton and that there is no  $(q, a, \sigma, \mathbf{u}, q') \in \delta_{\mathcal{A}}$  with  $q = q_f$ . Under this assumption and the fact that  $\mathcal{A}$  is deterministic we have that if  $q_0(0,0) \xrightarrow{*} q_f(0,0)$ , then there exists a unique run  $q_0(0,0) \xrightarrow{\rho} q_f(0,0)$ . Let  $Q^{(1)} = \{q^{(1)} \mid q \in Q_{\mathcal{A}}\}$  and  $Q^{(2)} = \{q^{(2)} \mid q \in Q_{\mathcal{A}}\}$  be fresh copies of  $Q_{\mathcal{A}}$ . We set  $Q = Q^{(1)} \uplus Q^{(2)} \uplus \{p^{(1)}, p^{(2)}\}$ , where  $p^{(1)}$  and  $p^{(2)}$  are two fresh control states. A *mode* is an element from  $(Q^{(1)} \cup Q^{(2)}) \times \{0,1\}$ .

The idea is that  $\mathcal{B}$ 's configuration in states  $Q^{(1)}$  are there to mimic the first counter of  $\mathcal{A}$ , whereas the configurations in states  $Q^{(2)}$  are there to mimic the second counter of  $\mathcal{A}$ . To this end, we define that a mode  $(q^{(i)}, \sigma) \in (Q^{(1)} \cup Q^{(2)}) \times \{0,1\}$  is *compatible* with a transition  $\tau = (r, a, \sigma, \mathbf{u}, r')$  if  $q = r$  and  $\sigma$  is the same signum as  $\tau$ 's signum of the counter that is to be mimicked, formally  $\sigma = \sigma(i)$ . Moreover, those configurations with control state in  $p^{(1)}$  or in  $p^{(2)}$  are denoted to be *erroneous*. Let us define the remaining components of  $\mathcal{B}$ .

We define the alphabet to be  $\Sigma = \delta_{\mathcal{A}} \uplus \{\#\}$ , where  $\#$  is a fresh symbol. The transitions are defined as  $\delta = \delta_{sim} \cup \delta_{restart} \cup \delta_{reset} \cup \delta_{err}$ , whose components we successively list and comment on below; in addition a general structure of the construction is presented in Figure 1.

The transitions in  $\delta_{sim}$  correspond to faithful simulations of transitions in  $\mathcal{A}$  in which, as already mentioned above, a state  $q^{(i)}$  is supposed to mimic the behavior on counter  $i$ :

$$\delta_{sim} = \{(q^{(i)}, \tau, \sigma(i), \mathbf{u}(i), r^{(i)}) \mid (q^{(i)}, \sigma(i)) \text{ is comp. with } \tau = (q, a, \sigma, \mathbf{u}, r), i \in \{1,2\}\}$$

The letter  $\#$  is to be understood as a letter to restart the simulation of the machine  $\mathcal{A}$  and is only treated as non-erroneous when executed from the configurations  $\{p^{(1)}(0), p^{(2)}(0)\}$  or from  $\{q_f^{(1)}(0), q_f^{(2)}(0)\}$ .

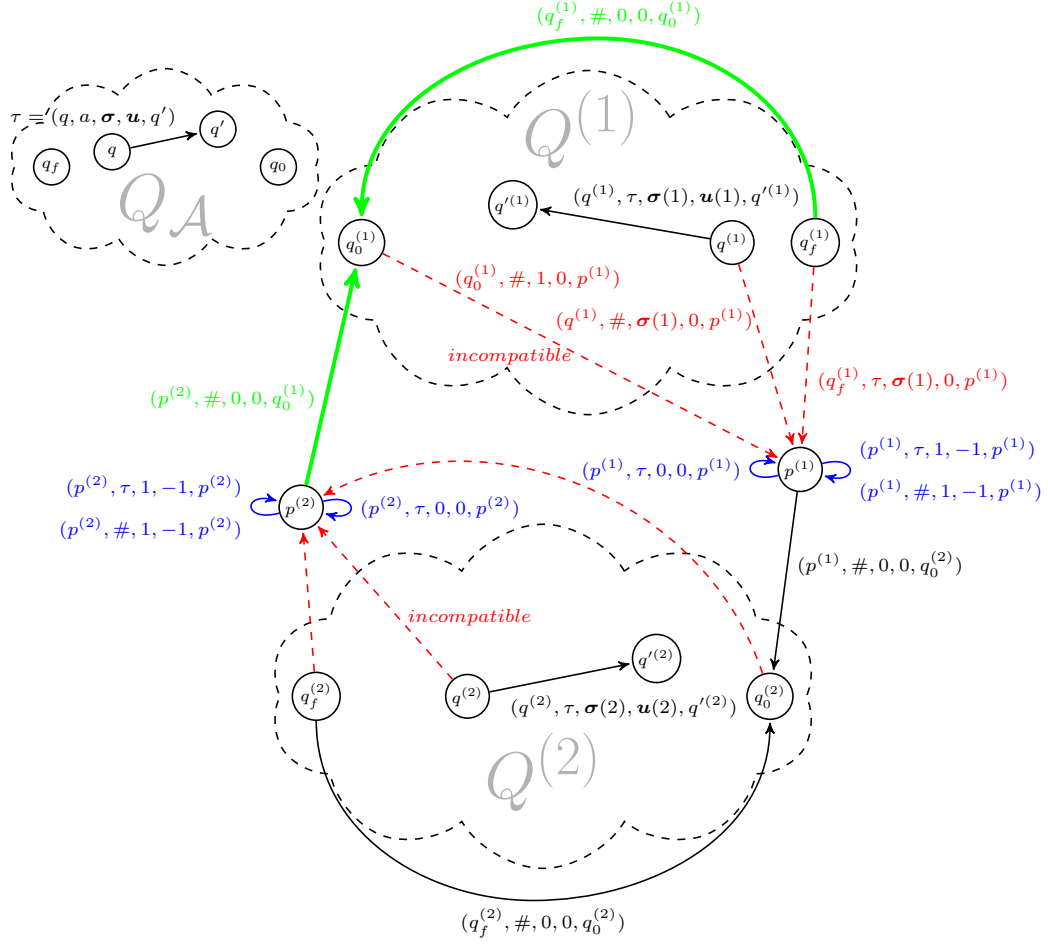
$$\delta_{restart} = \{(p^{(i)}, \#, 0, 0, q_0^{(3-i)}) \mid i \in \{1,2\}\} \cup \{(q_f^{(i)}, \#, 0, 0, q_0^{(i)}) \mid i \in \{1,2\}\} \quad (1)$$

The next type of transitions are transitions that are executed in order to reset the simulation, that is to lead back to a simulation of  $\mathcal{A}$  from configuration  $q_0(0,0)$ , possibly by decrementing the counter for it to eventually equal zero.

$$\delta_{reset} = \{(p^{(i)}, \tau, 1, -1, p^{(i)}), (p^{(i)}, \tau, 0, 0, p^{(i)}) \mid \tau \in \delta_{\mathcal{A}}, i \in \{1,2\}\} \quad (2)$$

$$\cup \{(p^{(i)}, \#, 1, -1, p^{(i)}) \mid i \in \{1,2\}\} \quad (3)$$

The last type of transitions that remain are erroneous transitions – they all lead to erroneous configurations. The first kind of erroneous transitions can be applied at configurations



■ **Figure 1** On the left hand side there is a cloud that is an emblematic description of the deterministic 2-counter automaton  $\mathcal{A}$  with the set of states  $Q_{\mathcal{A}}$ . On the right hand side the structure of the automaton  $\mathcal{B}$  is presented. Two clouds stand for two copies of  $Q_{\mathcal{A}}$ , where the bottom copy that represents  $Q^{(2)}$  is graphically flipped. Red dashed arrows are elements of  $\delta_{err}$ , blue arrows are elements of  $\delta_{reset}$ , thick green arrows are accepting.

of the form  $q^{(i)}(n) \in (Q^{(1)} \cup Q^{(2)}) \times \mathbb{N}$  whose mode  $(q^{(i)}, \text{sgn}(n))$  is incompatible with the letter  $\tau \in \delta_{\mathcal{A}}$  to be read. The second kind of erroneous transitions are transitions that read the symbol  $\#$  in a moment when a restart was not expected.

$$\delta_{err} = \{(q^{(i)}, \tau, \sigma, 0, p^{(i)}) \mid (q^{(i)}, \sigma) \text{ is incompatible with } \tau \in \delta_{\mathcal{A}}, i \in \{1, 2\}\} \quad (4)$$

$$\cup \{(q^{(i)}, \#, \sigma, 0, p^{(i)}) \mid q \in Q_{\mathcal{A}} \setminus \{q_f\}, \sigma \in \{0, 1\}, i \in \{1, 2\}\} \quad (5)$$

$$\cup \{(q_f^{(i)}, \#, 1, 0, p^{(i)}) \mid i \in \{1, 2\}\} \quad (6)$$

Since  $\mathcal{A}$  is deterministic it is readily checked that  $\mathcal{B}$  is deterministic as well. The set of final transitions is defined to be  $F = \{(q_f^{(1)}, \#, 0, 0, q_0^{(1)}), (p^{(2)}, \#, 0, 0, q_0^{(1)})\}$ .

For any two sets  $X$  and  $Y$  let  $\Delta(X, Y) = X \setminus Y \cup Y \setminus X$  denote their symmetric difference.

► **Lemma 4.** *If  $q_0(0, 0) \xrightarrow{*} q_f(0, 0)$ , then  $L_{\omega}(q_0^{(1)}(0)) \neq L_{\omega}(q_0^{(2)}(0))$ .*

**Proof.** Assume there exists a run  $q_0(0, 0) \xrightarrow{p} q_f(0, 0)$  in  $\mathcal{A}$ . Since  $\mathcal{B}$  is deterministic there exists a unique run  $q_0^{(1)}(0) \xrightarrow{\pi^{(1)}} q_f^{(1)}(0)$  and a unique run  $q_0^{(2)}(0) \xrightarrow{\pi^{(2)}} q_f^{(2)}(0)$  with  $\text{READ}(\pi^{(1)}) =$

$\text{READ}(\pi^{(2)}) = \rho$  in  $\mathcal{B}$ . Since  $\rho \in \delta_{\mathcal{A}}^*$  it follows  $\pi^{(1)}, \pi^{(2)} \in (\delta \setminus F)^*$ , i.e. neither  $\pi^{(1)}$  nor  $\pi^{(2)}$  contains any final transition. Moreover we have  $q_f^{(1)}(0) \xrightarrow{(q_f^{(1)}, \#, 0, 0, q_0^{(1)})} q_0^{(1)}(0)$  and  $q_f^{(2)}(0) \xrightarrow{(q_f^{(2)}, \#, 0, 0, q_0^{(2)})} q_0^{(2)}(0)$ , by construction of  $\mathcal{B}$ , where the former used transition is final and the latter used transition is not final.

Hence, on the infinite word  $(\rho\#)^\omega$  the unique  $\omega$ -run from  $q_0^{(1)}(0)$  encounters infinitely many final transitions, whereas the unique  $\omega$ -run from  $q_0^{(2)}(0)$  does not encounter any final transition. Thus  $(\rho\#)^\omega \in \Delta(L_\omega(q_0^{(1)}(0)), L_\omega(q_0^{(2)}(0)))$  and hence  $L_\omega(q_0^{(1)}(0)) \neq L_\omega(q_0^{(2)}(0))$ . ◀

Let us fix a sequence of transitions  $\pi \in \delta^\omega$ . For  $j \in \{1, 2\}$  we say  $\pi$  is  $j$ -pure if  $\pi[i] \in Q^{(j)} \times \Sigma \times \{0, 1\} \times \{-1, 0, 1\} \times Q^{(j)}$  for all  $i \geq 1$ .

► **Lemma 5.** *Let  $\pi$  be the  $\omega$ -run from some configuration in  $\mathcal{B}$  such that  $\#$  appears infinitely often in  $\text{READ}(\pi)$ . Then  $\pi$  is rejecting if, and only if,  $\pi[\ell, \infty]$  is 2-pure for some  $\ell \geq 1$ .*

**Proof.**

“If”: If  $\pi$  is 2-eventually, then clearly is  $\pi$  rejecting since every accepting transition has  $q_0^{(1)}$  as last component.

“Only-if”: Assume  $\pi$  is a rejecting  $\omega$ -run from the configuration  $q^{(i)}(n)$  in  $\mathcal{B}$ . Since  $\#$  appears infinitely often in  $\text{READ}(\pi)$  a simple inspection of the rules in  $\mathcal{B}$  (in particular the resetting behavior at states  $p^{(1)}$  and  $p^{(2)}$ , see also Figure 1) shows that  $\pi$  must contain infinitely many occurrences of at least one of the following transitions:

- $\tau_1 = (p^{(1)}, \#, 0, 0, q_0^{(2)})$
- $\tau_2 = (p^{(2)}, \#, 0, 0, q_0^{(1)})$
- $\tau_3 = (q_f^{(1)}, \#, 0, 0, q_0^{(1)})$
- $\tau_4 = (q_f^{(2)}, \#, 0, 0, q_0^{(2)})$

Surely, both  $\tau_2$  and  $\tau_3$  can only appear finitely often in  $\pi$  since  $F = \{\tau_2, \tau_3\}$  and  $\pi$  is assumed to be rejecting. Bearing in mind that  $\text{READ}(\pi)$  contains infinitely many  $\#$ 's one further observes that by construction of  $\mathcal{B}$  the transition  $\tau_1$  can only appear infinitely often if the transition  $\tau_2$  appears infinitely often. Therefore all the transitions  $\tau_1, \tau_2$  and  $\tau_3$  can only appear finitely often in  $\pi$ . Hence there exists some  $\ell \in \mathbb{N}$  such that  $\pi[\ell, \infty]$  contains neither of the transitions  $\tau_1, \tau_2$  nor  $\tau_3$ . Since  $\text{READ}(\pi[\ell, \infty])$  still contains infinitely many  $\#$ 's it follows that  $\pi[\ell, \infty]$  is 2-pure by construction of  $\mathcal{B}$ . ◀

Let  $\pi$  be a run or an  $\omega$ -run of  $\mathcal{B}$ . For any state  $r \in Q$ , we say that  $\pi$  is  $r$ -free if none of the transitions that appears in  $\pi$  contains  $r$  as first or as last component. The following lemma shows that in case an infinite word lies in the symmetric difference of the language of any two configurations of  $\mathcal{B}$ , then it eventually repeatedly simulates the unique run from  $q_0(0, 0)$  to  $q_f(0, 0)$  with a separating symbol  $\#$  in between, if such a run exists.

► **Lemma 6.** *Let  $s(m)$  and  $t(n)$  be two arbitrary configurations of  $\mathcal{B}$ . Then the following holds:*

1. *Every  $w \in \Delta(L_\omega(s(m)), L_\omega(t(n)))$  contains infinitely many  $\#$ 's.*
2. *Let  $w \in \Delta(L_\omega(s(m)), L_\omega(t(n)))$ , let  $\alpha$  be the unique run from  $s(m)$  and let  $\beta$  be the unique run from  $t(n)$  such that  $\text{READ}(\alpha) = \text{READ}(\beta) = w$ . Then there is some  $j \in \{1, 2\}$  such that*
  - (a)  $\alpha[h_1, \infty]$  *is  $j$ -pure and  $\beta[h_1, \infty]$  is  $(3 - j)$ -pure for some  $h_1 \geq 1$  and moreover*
  - (b)  $q_0(0, 0) \xrightarrow{*} q_f(0, 0)$  *and  $w \in \Sigma^*(\rho\#)^\omega$ , where  $q_0(0, 0) \xrightarrow{\rho} q_f(0, 0)$  is the unique run from  $q_0(0, 0)$  to  $q_f(0, 0)$ .*

**Proof.** Point (1) immediately follows from the fact that  $\text{READ}(\tau) = \#$  for all  $\tau \in F$ .

Let us show (2). Let us fix an arbitrary  $w \in \Delta(L_\omega(s(m)), L_\omega(t(n)))$ . Let  $s(m) \xrightarrow{\alpha}$  be the unique run from  $s(m)$  and let  $t(n) \xrightarrow{\beta}$  be the unique run from  $t(n)$  such that  $\text{READ}(\alpha) = \text{READ}(\beta) = w$ . Without loss of generality let us assume that  $\alpha$  is accepting and that  $\beta$  is rejecting. Then there exists a position  $h_0 \in \mathbb{N}$  such that

$$\beta[h_0, \infty] \text{ is 2-pure, due to Lemma 5.} \quad (7)$$

This in turn, by construction of  $\mathcal{B}$  and recalling that any transition having  $p^{(2)}$  as source or target state is *not* 2-pure, implies that every transition in  $\beta$  that reads a  $\#$  must lead to the configuration  $q_0^{(2)}(0)$ , i.e.:

$$\forall i \geq h_0 : (\text{READ}(\beta[i]) = \# \implies \beta[i] = (q_f^{(2)}, \#, 0, 0, q_0^{(2)})) \quad (8)$$

For establishing (2.a), it suffices to show that  $\alpha[h_1, \infty]$  is 1-pure for some  $h_1 \geq h_0$ . For this, we prove the following claim, whose proof we postpone to the end.

► **Claim 7.**  $\alpha[h_0, \infty]$  is  $p^{(1)}$ -free.

We will show that  $\alpha[h_1, \infty]$  is 1-pure for some  $h_1 \geq h_0$ . As  $\alpha$  is accepting and  $\alpha[h_0, \infty]$  is  $p^{(1)}$ -free the only possible accepting transition that appears infinitely often in  $\alpha$  must be the transition  $(q_f^{(1)}, \#, 0, 0, q_0^{(1)})$ . Hence there exists some  $h_1 \geq h_0$  such that  $\alpha[h_1, \infty]$  does not contain the other accepting transition, namely  $(p^{(2)}, \#, 0, 0, q_0^{(1)})$ . Altogether we have that the only transition labeled with  $\#$  in  $\alpha[h_1, \infty]$  is  $(q_f^{(1)}, \#, 0, 0, q_0^{(1)})$ . So  $\alpha[h_1, \infty]$  is 1-pure which immediately follows from construction of  $\mathcal{B}$  and from the fact that  $\text{READ}(\alpha[h_1, \infty])$  contains infinitely many  $\#$ 's. This shows (2.a.).

Let us finally prove (2b.). Still, the word  $w[h_1, \infty]$  contains infinitely many  $\#$ 's. That is, we can uniquely factorize  $w[h_1, \infty]$  as

$$w[h_1, \infty] = w_0 \# w_1 \# \dots$$

where  $w_\ell \in \delta_{\mathcal{A}}^*$  for each  $\ell \geq 0$ . Since  $\alpha[h_1, \infty]$  is 1-pure and therefore  $p^{(1)}$ -free and  $\beta[h_1, \infty]$  is 2-pure and therefore  $p^{(2)}$ -free it follows from construction of  $\mathcal{B}$  that for all  $\ell \geq 0$ :

- $s(m) \xrightarrow{w[1, h_1 - 1] w_1 \# w_2 \dots \# w_\ell} q_f^{(1)}(0) \xrightarrow{\#} q_0^{(1)}(0)$  and
- $t(n) \xrightarrow{w[1, h_1 - 1] w_1 \# w_2 \dots \# w_\ell} q_f^{(2)}(0) \xrightarrow{\#} q_0^{(2)}(0)$ .

Therefore

$$\forall \ell \geq 1 : q_0^{(1)}(0) \xrightarrow{w_\ell} q_f^{(1)}(0) \quad \text{and} \quad q_0^{(2)}(0) \xrightarrow{w_\ell} q_f^{(2)}(0) \quad . \quad (9)$$

Recall that  $\text{READ}(\alpha[h_1, \infty]) = \text{READ}(\beta[h_1, \infty]) = w_1 \# w_2 \# \dots$ . Again since  $\alpha[h_1, \infty]$  is 1-pure and  $\beta[h_1, \infty]$  is 2-pure we can conclude from (9) and the construction of  $\mathcal{B}$  that

$$\forall \ell \geq 1 : q_0(0, 0) \xrightarrow{w_\ell} q_f(0, 0) \quad .$$

But since  $\mathcal{A}$  is assumed to be deterministic and the latter is clearly only possible when  $q_0(0, 0) \xrightarrow{*} q_f(0, 0)$  we must therefore have  $w_\ell = \rho$  for all  $\ell \geq 1$ , where  $\rho$  denotes the unique run from  $q_0(0, 0)$  to  $q_f(0, 0)$ . Hence Point (2b.) follows.

**Proof of the Claim.** Assume by contradiction that  $\alpha[h_0, \infty]$  is not  $p^{(1)}$ -free. Then there exists some  $k \geq h_0$  such that  $\alpha[k] = (q, x, \sigma, u, p^{(1)}) \in \delta$   $\alpha[k] = (p^{(1)}, x, \sigma, u, q) \in \delta$  for some  $q \in Q$ , some  $x \in \Sigma$ , some  $\sigma \in \{0, 1\}$ , and some  $u \in \{-1, 0, 1\}$ . We only treat the case  $\alpha[k] = (q, x, \sigma, u, p^{(1)}) \in \delta$  here, the other case is analogous. Hence,  $s(m) \xrightarrow{\alpha[1, k]} p^{(1)}(m_1)$  for some  $m_1 \in \mathbb{N}$ . In the same moment (recalling that  $\beta[h_0, \infty]$  is 2-pure) we have  $t(n) \xrightarrow{\beta[1, k]} q^{(2)}(n_1)$



for some  $n_1 \in \mathbb{N}$  and some  $q^{(2)} \in Q^{(2)}$ . Of course  $w[k+1, \infty]$  still contains infinitely many  $\#$ 's. Let  $z$  be the shortest finite prefix of  $w[k+1, \infty]$  of length at least  $m_1 + 1$  that ends with a  $\#$ . Note that  $w[1, k]z$  is a finite prefix of our infinite word  $w$ .

We have  $p^{(1)}(m_1) \xrightarrow{z} q_0^{(2)}(0)$  by definition of  $z$  and the construction of  $\mathcal{B}$  and therefore  $s(m) \xrightarrow{w[1, k]z} q_0^{(2)}(0)$ . Moreover, since  $z$  ends with a  $\#$  we have  $t(n) \xrightarrow{w[1, k]z} q^{(2)}(0)$  by (8). Thus, both  $s(m) \xrightarrow{w[1, k]z} q_0^{(2)}(0)$  and  $t(n) \xrightarrow{w[1, k]z} q^{(2)}(0)$ . Since  $\mathcal{B}$  is deterministic this clearly contradicts  $w \in \Delta(L_\omega(s(m)), L_\omega(t(n)))$  and ends the proof of the claim.  $\blacktriangleleft$

Lemma 4 and Lemma 6 together imply that  $q_0(0, 0) \xrightarrow{*} q_f(0, 0)$  if, and only if,  $L_\omega(q_0^{(1)}(0)) \neq L_\omega(q_0^{(2)}(0))$ . Hence Theorem 2 follows.

#### 4 The universality problem for 1-counter nets

In this section we will work with  $k$ -counter automata that have incremental errors. For this it is convenient to denote for each  $i \in [1, k]$  the unit vector  $e_i \in \mathbb{N}^k$ , i.e.  $e_i(i) = 1$  and  $e_i(j) = 0$  for all  $j \in [1, k] \setminus \{i\}$ . We define  $E = \{e_i \mid i \in [1, k]\}$ . Given a  $k$ -counter automaton  $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma_{\mathcal{A}}, \delta_{\mathcal{A}}, q_0, F_{\mathcal{A}})$  we define an *incrementing-error transition relation*  $\xrightarrow{\tau}_+$  as follows, where  $\tau$  ranges over  $\delta_{\mathcal{A}} \cup E$  and  $p(\mathbf{m})$  and  $q(\mathbf{n})$  are configurations:

$$p(\mathbf{m}) \xrightarrow{\tau}_+ q(\mathbf{n}) \text{ if } \begin{cases} \tau \in \delta_{\mathcal{A}} \text{ and } p(\mathbf{m}) \xrightarrow{a} q(\mathbf{n}), \text{ or} \\ \tau = e_i \text{ and } p = q \text{ and } \mathbf{n} = \mathbf{m} + e_i \end{cases}$$

The relation  $\xrightarrow{\tau}_+$  is naturally extended to words over  $\delta_{\mathcal{A}} \cup E$ . An *inc-run* is a run of the form  $p(\mathbf{m}) \xrightarrow{\rho}_+ q(\mathbf{n})$  for some  $\rho \in (\delta_{\mathcal{A}} \cup E)^*$ . We say that  $q(\mathbf{n})$  is *inc-reachable* from  $p(\mathbf{m})$  (and also write  $p(\mathbf{m}) \xrightarrow{*}_+ q(\mathbf{n})$  for this) if  $p(\mathbf{m}) \xrightarrow{\rho}_+ q(\mathbf{n})$  for some  $\rho \in (\delta_{\mathcal{A}} \cup E)^*$ . Finally, for a word  $w \in (\Sigma_{\mathcal{A}} \cup E)^*$  we write  $p(\mathbf{m}) \xrightarrow{w}_+ q(\mathbf{n})$  if there is  $\rho$  an inc-run  $p(\mathbf{m}) \xrightarrow{\rho}_+ q(\mathbf{n})$  such that  $\text{READ}(\rho) = w$  where  $\text{READ}$  is defined like in Section 2 and stipulating  $\text{READ}(e_i) = e_i$  for every  $1 \leq i \leq k$ .

Let  $q_f \in Q$ . We write  $q_0(\mathbb{N}, 0, \dots, 0) \xrightarrow{*}_+ q_f(\mathbf{0})$  if for all  $n \in \mathbb{N}$  we have  $q_0(n, 0, \dots, 0) \xrightarrow{*}_+ q_f(\mathbf{0})$ . We also say that  $q_f$  is *forall inc-reachable* from  $q_0$ .

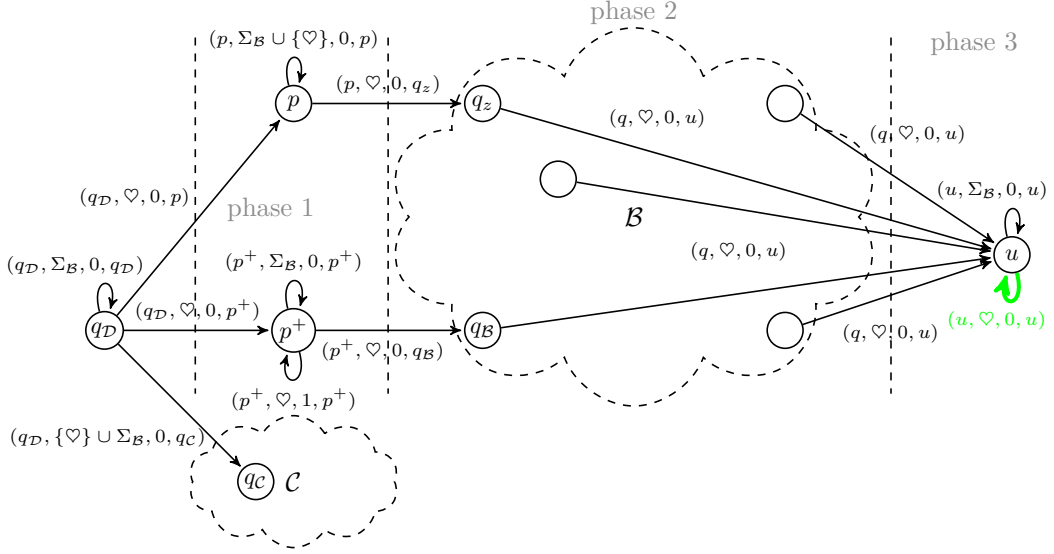
The following lemma can be proven by reduction from the boundedness problem for decremental-error  $k$ -counter automata (a.k.a. lossy counter machines) [16].

► **Lemma 8.** *Given a  $k$ -counter automaton and two of its states  $q_0$  and  $q_f$ , the question if  $q_f$  is forall inc-reachable from  $q_0$  is undecidable.*

The following lemma states that one can simulate inc-runs of  $k$ -counter automata in terms of a universality problem for 1-counter nets. The construction was essentially already present in Theorem 3 in [7], however we need a more general simulation lemma since we are interested in simulating runs that start in particular configurations (and not in configurations that have all its counters zero).

► **Lemma 9.** *For a given  $k$ -counter automaton  $\mathcal{A} = (Q_{\mathcal{A}}, \Sigma_{\mathcal{A}}, \delta_{\mathcal{A}}, q_0)$  and a state  $q_f \in Q_{\mathcal{A}}$  one can construct a 1-counter net  $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma_{\mathcal{B}}, \delta_{\mathcal{B}}, q_{\mathcal{B}}, \delta_{\mathcal{B}})$  with a state  $q_z \in Q_{\mathcal{B}}$  such that for every  $n \in \mathbb{N}$  the two following statements are equivalent.*

- $q_0(n, 0 \dots 0) \xrightarrow{*}_+ q_f(\mathbf{0})$  in  $\mathcal{A}$ .
- $L(\mathcal{B}, q_{\mathcal{B}}(n)) \cup L(\mathcal{B}, q_z(0))$  is not universal.



■ **Figure 2** Picture of the one counter net  $\mathcal{D}$ . The green thick transitions are accepting (the accepting transitions in  $\mathcal{C}$  are hidden). Transitions that have a set of letters instead of a single letter, like  $(q_{\mathcal{D}}, \Sigma_{\mathcal{B}}, 0, q_{\mathcal{D}})$ , denotes a set of transitions, one for every element of the set of letters.

▶ **Remark.** Note that all transitions in the constructed net  $\mathcal{B}$  are accepting. Thus, if a word  $w$  does not belong to the language  $L(\mathcal{B}, q_z(0)) \cup L(\mathcal{B}, q_B(n))$ , then there is no run for the word  $w$  in  $\mathcal{B}$  that starts in  $q_z(0)$  or in  $q_B(n)$ . This will be used in the proof of Claim 12.

Lemma 8 and Lemma 9 immediately imply the following undecidability result.

▶ **Corollary 10.** *Given a 1-counter net  $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma_{\mathcal{B}}, \delta_{\mathcal{B}}, q_{\mathcal{B}}, \delta_{\mathcal{B}})$  and a state  $q_z \in Q_{\mathcal{B}}$  it is undecidable if for all  $n \in \mathbb{N}$  we have that  $L(\mathcal{B}, q_B(n)) \cup L(\mathcal{B}, q_z(0))$  is not universal.*

#### 4.1 Proof of Theorem 3

We reduce the problem proven to be undecidable in Corollary 10 to  $\omega$ -UNIVERSALITY of 1-counter nets.

Formally, for a given 1-counter net  $\mathcal{B} = (Q_{\mathcal{B}}, \Sigma_{\mathcal{B}}, \delta_{\mathcal{B}}, q_{\mathcal{B}}, \delta_{\mathcal{B}})$  we construct a 1-counter net  $\mathcal{D} = (Q_{\mathcal{D}}, \Sigma_{\mathcal{D}}, \delta_{\mathcal{D}}, q_{\mathcal{D}}, F_{\mathcal{D}})$  such that the  $\omega$ -language  $L_{\omega}(\mathcal{D}, q_{\mathcal{D}}(0))$  is universal if, and only if,  $L(\mathcal{B}, q_B(n)) \cup L(\mathcal{B}, q_z(0))$  is universal for some  $n \in \mathbb{N}$ .

We first define the following gadget: given the finite alphabet  $\Sigma_{\mathcal{B}}$  and a symbol  $\heartsuit \notin \Sigma_{\mathcal{B}}$  we construct the Büchi 1-counter net  $\mathcal{C} = (Q_{\mathcal{C}}, \Sigma_{\mathcal{B}} \cup \{\heartsuit\}, \delta_{\mathcal{C}}, q_{\mathcal{C}}, F_{\mathcal{C}})$  such that  $L_{\omega}(\mathcal{C}, q_{\mathcal{C}}(0)) = (\Sigma_{\mathcal{B}} \cup \{\heartsuit\})^* \Sigma_{\mathcal{B}}^{\omega}$ , which is in fact even an  $\omega$ -regular language.

Define  $\mathcal{C}$  as follows:

$$\begin{aligned}
 Q_{\mathcal{C}} &= \{q_{\mathcal{C}}, q_1\} \\
 \delta_{\mathcal{C}} &= \{(q_{\mathcal{C}}, a, 0, q_{\mathcal{C}}), (q_{\mathcal{C}}, a, 0, q_1), (q_1, a, 0, q_1) \mid a \in \Sigma_{\mathcal{B}}\} \cup \{(q_{\mathcal{C}}, \heartsuit, 0, q_{\mathcal{C}})\} \\
 F_{\mathcal{C}} &= \{(q_1, a, 0, q_1) \mid a \in \Sigma_{\mathcal{B}}\}.
 \end{aligned}$$

It is easy to check that the language accepted by  $\mathcal{C}$  is indeed  $(\Sigma_{\mathcal{B}}^* \heartsuit)^* \Sigma_{\mathcal{B}}^{\omega}$ .

Figure 2 depicts the construction of  $\mathcal{D}$ . The idea is that we introduce a special symbol  $\heartsuit$  such that the only relevant words are those that contain infinitely many  $\heartsuit$ 's. We filter out all words with finite number of  $\heartsuit$  using the gadget  $\mathcal{C}$ . Next, the net  $\mathcal{D}$  has three phases between

which we switch reading the symbol  $\heartsuit$ . The first phase is responsible for setting up the inputs to the net  $\mathcal{B}$ . It reads  $\heartsuit$ 's and nondeterministically decides to increment the counter or go to the second phase. The second phase tests if  $L(\mathcal{B}, q_{\mathcal{B}}(n)) \cup L(\mathcal{B}, q_z(0))$  is not universal for a given  $n$ . The third phase is to accept if it happens that for a given  $n$  the language  $L(\mathcal{B}, q_{\mathcal{B}}(n)) \cup L(\mathcal{B}, q_z(0))$  is universal. Now observe that if there is some  $n \in \mathbb{N}$  such that  $L(\mathcal{B}, q_{\mathcal{B}}(n)) \cup L(\mathcal{B}, q_z(0))$  is universal, then for every run with infinitely many occurrences of  $\heartsuit$  we can increment the counter to the value  $n$  then go to the second phase in which we cannot get stuck due to the universality of  $L(\mathcal{B}, q_{\mathcal{B}}(n)) \cup L(\mathcal{B}, q_z(0))$ . Finally, seeing the next  $\heartsuit$  we jump to the third phase where we accept. Conversely, if for all  $n \in \mathbb{N}$  we have  $L(\mathcal{B}, q_{\mathcal{B}}(n)) \cup L(\mathcal{B}, q_z(0))$  is not universal, then one can build an  $\omega$ -word such that no matter how long we wait in the first phase incrementing the counter, when we decide to jump to the second phase we will get stuck in it before having the possibility to jump to phase 3; so there is no possibility of accepting this specially designed word.

Formally, the 1-counter net  $\mathcal{D}$  is defined as follows:

$$\begin{aligned}
Q_{\mathcal{D}} &= \{q_{\mathcal{D}}, p, p^+, u\} \cup Q_{\mathcal{C}} \cup Q_{\mathcal{B}} \\
\Sigma_{\mathcal{D}} &= \Sigma_{\mathcal{B}} \cup \{\heartsuit\} \\
\delta_{\mathcal{D}} &= \delta_{\mathcal{C}} \cup \delta_{\mathcal{B}} \\
&\cup \{(u, a, 0, u), (p, a, 0, p), (p^+, a, 0, p^+), (q_{\mathcal{D}}, a, 0, q_{\mathcal{D}}) \mid a \in \Sigma_{\mathcal{B}}\} \\
&\cup \{(u, \heartsuit, 0, u), (p, \heartsuit, 0, p), (p^+, \heartsuit, 1, p^+)\} \\
&\cup \{(q_{\mathcal{D}}, \heartsuit, 0, p), (q_{\mathcal{D}}, \heartsuit, 0, p^+), (q_{\mathcal{D}}, \heartsuit, 0, q_{\mathcal{C}})\} \\
&\cup \{(p, \heartsuit, 0, q_z), (p^+, \heartsuit, 0, q_{\mathcal{B}})\} \\
&\cup \{(q, \heartsuit, 0, u) \mid q \in Q_{\mathcal{B}}\} \\
F_{\mathcal{D}} &= F_{\mathcal{C}} \cup \{(u, \heartsuit, 0, u)\}
\end{aligned}$$

We start with a general observation on  $\mathcal{D}$ . The language  $L_{\omega}(\mathcal{D}, q_{\mathcal{D}}(0))$  can be partitioned into two sets, each of them being accepted by a different subset of the set of accepting transitions.

1. The set of words in which the letter  $\heartsuit$  appears finitely often. They are indeed all accepted by  $\mathcal{D}$ , namely by runs in which one of the transitions  $(q_{\mathcal{D}}, \heartsuit, 0, q_{\mathcal{C}})$  or  $(q_{\mathcal{D}}, a, 0, q_{\mathcal{C}})$  for  $a \in \Sigma_{\mathcal{B}}$  is used, finally allowing transitions in  $F_{\mathcal{C}}$  to be used infinitely often.
2. The set of words in which the letter  $\heartsuit$  appears infinitely often. In this case acceptance has to be due to an infinite number of occurrences of the transition  $(u, \heartsuit, 0, u)$ .

Due to case 1 universality of  $L_{\omega}(\mathcal{D}, q_{\mathcal{D}}(0))$  holds if, and only if,  $(\Sigma_{\mathcal{B}}^* \heartsuit)^{\omega} \subseteq L_{\omega}(\mathcal{D}, q_{\mathcal{D}}(0))$ .

► **Claim 11.** *If  $L(\mathcal{B}, q_{\mathcal{B}}(n)) \cup L(\mathcal{B}, q_z(0))$  is universal for some  $n \in \mathbb{N}$ , then  $(\Sigma_{\mathcal{B}}^* \heartsuit)^{\omega} \subseteq L_{\omega}(\mathcal{D}, q_{\mathcal{D}}(0))$ .*

Let  $w \in (\Sigma_{\mathcal{B}}^* \heartsuit)^{\omega}$ , i.e.  $w$  contains infinitely many  $\heartsuit$ 's. Let  $w'$  be the infix of  $w$  that is strictly in between the  $(n+2)$ -th and  $(n+3)$ -th occurrence of the letter  $\heartsuit$ . Due to our assumption  $w'$  belongs to  $L(\mathcal{B}, q_z(0))$  or to  $L(\mathcal{B}, q_{\mathcal{B}}(n))$ .

If  $w' \in L(\mathcal{B}, q_z(0))$ , then  $w$  is accepted by the following run. We wait for the first  $\heartsuit$  in state  $q_{\mathcal{D}}$  and then we use transition  $(q_{\mathcal{D}}, \heartsuit, 0, p)$ . Next, we loop in the state  $p$  until the  $(n+2)$ -th occurrence of the letter  $\heartsuit$  on which we move to the configuration  $q_z(0)$  using the transition  $(p, \heartsuit, 0, q_z)$ . At this point, we follow an accepting run in  $\mathcal{B}$  starting from  $q_z(0)$  for the word  $w'$  and then on the next occurrence of  $\heartsuit$  we move to state  $u$  via the transition  $(q, \heartsuit, 0, u)$  for some  $q \in Q_{\mathcal{B}}$ . Finally, in  $u$  we accept by performing the transition  $(u, \heartsuit, 0, u)$  infinitely often.

If  $w' \in L(\mathcal{B}, q_{\mathcal{B}}(n))$ , then  $w$  is accepted by the following run. We wait for the first occurrence of  $\heartsuit$  in state  $q_{\mathcal{D}}$ . Next we use transition  $(q_{\mathcal{D}}, \heartsuit, 0, p^+)$ , then we stay in state  $p^+$  just before the  $(n+2)$ -th occurrence of  $\heartsuit$ , increasing the counter value to  $n$ . Now on upon reading  $\heartsuit$  we move to the configuration  $q_{\mathcal{B}}(n)$  using transition  $(p^+, \heartsuit, 0, q_{\mathcal{B}})$ . At this moment we follow an accepting run in  $\mathcal{B}$  starting from  $q_{\mathcal{B}}(n)$  for the word  $w'$  and then on the next occurrence of  $\heartsuit$  we move to state  $u$  via the transition  $(q, \heartsuit, 0, u)$  for some  $q \in Q_{\mathcal{B}}$ . Finally, in  $u$  we accept performing transition  $(u, \heartsuit, 0, u)$  infinitely often. This finishes the proof of the claim.

► **Claim 12.** *If  $L(\mathcal{B}, q_{\mathcal{B}}(n)) \cup L(\mathcal{B}, q_z(0))$  is not universal for all  $n \in \mathbb{N}$ , then  $(\Sigma_{\mathcal{B}}^* \heartsuit)^\omega \not\subseteq L_\omega(\mathcal{D}, q_{\mathcal{D}}(0))$ .*

First, observe for every  $n \in \mathbb{N}$  there is a finite word  $w_n \in \Sigma_{\mathcal{B}}^*$  such that  $w_n \notin L(\mathcal{B}, q_z(0)) \cup L(\mathcal{B}, q_{\mathcal{B}}(n))$ .

Let  $v_\omega = \heartsuit w_0 \heartsuit w_1 \heartsuit w_2 \heartsuit \dots$ . Since  $v_\omega$  contains infinitely many  $\heartsuit$ 's, it can only be accepted by infinitely many transitions of the form  $(u, \heartsuit, 0, u)$ , due to construction of  $\mathcal{D}$ . Moreover, a run using the transition  $(u, \heartsuit, 0, u)$  infinitely often must exactly once use either  $(p, \heartsuit, 0, q_{\mathcal{B}})$  or  $(p^+, \heartsuit, 0, q_z)$ . Each of these two transitions read the letter  $\heartsuit$ . For the sake of contradiction, suppose  $v_\omega \in L_\omega(\mathcal{D}, q_{\mathcal{D}}(0))$  is accepted by the run  $\rho$ . We split  $\rho$  into three parts  $\rho = \rho_0 \rho_1 \rho_2$ , where  $\rho_0$  is the longest prefix of  $\rho$  that does not contain  $(p, \heartsuit, 0, q_{\mathcal{B}})$  or  $(p^+, \heartsuit, 0, q_z)$ ,  $\rho_1$  is either of the latter two transitions (length one), and  $\rho_2$  is a remaining infinite suffix. Suppose, the symbol  $\heartsuit$  appears  $(i+2)$  times in  $\text{READ}(\rho_0)$ , then the configuration of the net  $\mathcal{B}$  after reading  $\rho_0$  can be  $p(0)$  or  $p^+(i)$ . So after reading  $\rho_0 \rho_1$  the run has reached either  $q_z(0)$  or  $q_{\mathcal{B}}(i)$ . Moreover we have  $\text{READ}(\rho_2) = w_i \heartsuit w_{i+1} \dots$ . However, we know that there is no run for word  $w_i$  in the net  $\mathcal{B}$ , neither one starting from  $q_z(0)$  nor one starting from  $q_{\mathcal{B}}(i)$ , thus  $\rho_2$  cannot be a run in  $\mathcal{D}$  starting from  $q_z(0)$  or  $q_{\mathcal{B}}(i)$ . Concluding, we have established a contradiction to the fact that  $\rho$  was an accepting run. Thus,  $v_\omega \in (\Sigma_{\mathcal{B}}^* \heartsuit)^\omega \setminus L_\omega(\mathcal{D}, q_{\mathcal{D}}(0))$ .

## 5 Conclusion and outlook

In this paper we have shown that the following two problems are undecidable on infinite words although being decidable finite words. First, we showed that the equivalence problem of deterministic Büchi 1-counter automata is undecidable (Theorem 2). Second, we have shown that the universality problem for Büchi 1-counter nets is undecidable (Theorem 3) by a sequence of reductions from the boundedness problem for incremental-error  $k$ -counter machines.

The exact recursive complexity of both problems is yet unclear to us and is planned to be investigated in the full version of this paper.

---

### References

- 1 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. of STOC*, pages 202–211. ACM, 2004.
- 2 Stanislav Böhm, Stefan Göller, and Petr Jančár. Equivalence of deterministic one-counter automata is NL-complete. In *Proc. of STOC*, pages 131–140, Palo Alto, California, USA, June 2013. ACM Press. URL: <http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/BGJ-stoc13.pdf>, doi:10.1145/2488608.2488626.
- 3 Didier Caucal, Dung T. Huynh, and Lu Tian. Deciding Branching Bimilarity of Normed Context-Free Processes Is in  $\Sigma_2^P$ . *Inf. Comput.*, 118(2):306–315, 1995.

- 4 Yuxi Fu. Checking Equality and Regularity for Normed BPA with Silent Moves. In *Proc. of ICALP*, Lecture Notes in Computer Science. Springer, 2013. to appear.
- 5 Yoram Hirshfeld and Mark Jerrum. Bisimulation equivalance is decidable for normed process algebra. In *Proc. of ICALP*, volume 1644 of *Lecture Notes in Computer Science*, pages 412–421. Springer, 1999.
- 6 Yoram Hirshfeld, Mark Jerrum, and Faron Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science*, 158(1&2):143–159, 1996.
- 7 Piotr Hofman and Patrick Totzke. Trace inclusion for one-counter nets revisited. In *Proc. of Reachability Problems*, pages 151–162, 2014.
- 8 Petr Jancar. Equivalences of pushdown systems are hard. In *Foundations of Software Science and Computation Structures – 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, pages 1–28, 2014. doi:10.1007/978-3-642-54830-7\_1.
- 9 Christof Löding. Simplification Problems for Deterministic Pushdown Automata on Infinite Words. *Int. J. Found. Comput. Sci.*, 26(8):1041–1068, 2015.
- 10 Christof Löding and Stefan Repke. Regularity Problems for Weak Pushdown  $\omega$ -Automata and Games. In *In Proc. of MFCS*, pages 764–776. Springer, 2012.
- 11 M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall International, Englewood Cliffs, 1967.
- 12 Dirk Nowotka and Jiri Srba. Height-Deterministic Pushdown Automata. In *Proc. of MFCS*, volume 4708 of *Lecture Notes in Computer Science*, pages 125–134. Springer, 2007.
- 13 Joël Ouaknine and James Worrell. On Metric Temporal Logic and Faulty Turing Machines. In *In Proc. of FoSSaCSs*, volume 3921 of *Lecture Notes in Computer Science*, pages 217–230. Springer, 2006.
- 14 Joël Ouaknine and James Worrell. On the decidability and complexity of Metric Temporal Logic over finite words. *Logical Methods in Computer Science*, 3(1), 2007.
- 15 S. Repke. *Simplification problems for automata and games*. PhD thesis, RWTH Aachen, 2014.
- 16 Philippe Schnoebelen. Lossy Counter Machines Decidability Cheat Sheet. In Antonín Kučera and Igor Potapov, editors, *Proc. of Reachability Problems*, pages 51–75. Springer, 2010.
- 17 Géraud Sénizergues.  $L(A)=L(B)$ ? decidability results from complete formal systems. *Theor. Comput. Sci.*, 251(1-2):1–166, 2001.
- 18 Colin Stirling. Deciding DPDA Equivalence Is Primitive Recursive. In *Proc. of ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 821–832. Springer, 2002.
- 19 Leslie G. Valiant. Regularity and related problems for deterministic pushdown automata. *J. ACM*, 22(1):1–10, January 1975.



# Optimizing Tree Decompositions in MSO<sup>\*†</sup>

Mikołaj Bojańczyk<sup>1</sup> and Michał Pilipczuk<sup>2</sup>

1 Institute of Informatics, University of Warsaw, Warsaw, Poland  
bojan@mimuw.edu.pl

2 Institute of Informatics, University of Warsaw, Warsaw, Poland  
michal.pilipczuk@mimuw.edu.pl

---

## Abstract

The classic algorithm of Bodlaender and Kloks [J. Algorithms, 1996] solves the following problem in linear fixed-parameter time: given a tree decomposition of a graph of (possibly suboptimal) width  $k$ , compute an optimum-width tree decomposition of the graph. In this work, we prove that this problem can also be solved in MSO in the following sense: for every positive integer  $k$ , there is an MSO transduction from tree decompositions of width  $k$  to tree decompositions of optimum width. Together with our recent results [LICS 2016], this implies that for every  $k$  there exists an MSO transduction which inputs a graph of treewidth  $k$ , and nondeterministically outputs its tree decomposition of optimum width.

**1998 ACM Subject Classification** F.4.3 Formal Languages, G.2.2 Graph Theory

**Keywords and phrases** tree decomposition, treewidth, transduction, monadic second-order logic

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.15

## 1 Introduction

Consider the following problem: given a tree decomposition of a graph of some width  $k$ , possibly suboptimal, we would like to compute an optimum-width tree decomposition of the graph. A classic algorithm of Bodlaender and Kloks [4] solves this problem in linear fixed-parameter time complexity, where the input width  $k$  is the parameter.

► **Theorem 1** (Bodlaender and Kloks, [4]). *There exists an algorithm that, given a graph  $G$  on  $n$  vertices and its tree decomposition of width  $k$ , runs in time  $2^{\mathcal{O}(k^3)} \cdot n$  and returns a tree decomposition of  $G$  of optimum width.*

The algorithm of Bodlaender and Kloks proceeds by a bottom-up dynamic programming procedure on the input decomposition. For every subtree, a set of partial optimum-width decompositions is computed. The crucial ingredient is a combinatorial analysis of partial decompositions which shows that only some small subset of them, of size bounded only by a function of  $k$ , needs to be remembered for future computation.

The algorithm of Bodlaender and Kloks is a key subroutine in the linear-time algorithm for computing the treewidth of a graph, due to Bodlaender [2]. The fact that the algorithm is essentially governed by a run of a finite-state automaton on the input tree decomposition was also used in the recent approximation algorithm of Bodlaender et al. [3]. The notion of

---

\* A full version of the paper is available at <https://arxiv.org/abs/1701.06937>.

† The research of M. Bojańczyk is supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant LIPA, agreement no. 683080). Mi. Pilipczuk is supported by the Foundation for Polish Science via the START stipend programme.



*typical sequences*, which is the main component of the analysis of partial decompositions, has found applications in algorithms for computing other width measures, like branchwidth [5], cutwidth [16, 17], or pathwidth of matroids [12]. The concept of typical sequences originates in the previous work of Courcelle and Lagergren [8] and of Lagergren and Arnborg [15].

**Our results.** The main result of this paper (Theorem 2) is that the problem of Bodlaender and Kloks can be solved by an MSO *transduction*, which is a way of describing nondeterministic transformations of relational structures using monadic second-order logic. More precisely, we show that for every  $k \in \{0, 1, 2, \dots\}$  there is an MSO transduction that inputs a tree decomposition of width  $k$  of a graph  $G$ , and outputs nondeterministically a tree decomposition of  $G$  of optimum width.

As a corollary of our main result, we show (Corollary 3) that an MSO transduction can compute an optimum-width tree decomposition, even if the input is only the graph and not a (possibly suboptimal) tree decomposition. This application is obtained by combining the main result of this paper with Theorem 2.4 from [6], which says that for every  $k \in \{0, 1, 2, \dots\}$  there is an MSO transduction which inputs a graph of treewidth  $k$  and outputs nondeterministically one of its tree decompositions of possibly suboptimal width at most  $f(k)$ , for some function  $f$ . In particular, we thus strengthen Theorem 2.4 of [6] by making the output a decomposition of exactly the optimum width, instead of only bounded by a function of the optimum.

Our proof is divided into a few steps. First, we prove a result called the *Dealternation Lemma*, which shows that there always exists an optimum-width tree decomposition that has bounded “alternation” with respect to the input suboptimal decomposition. Intuitively, small alternation is the key property allowing an optimum-width tree decomposition to be captured by an MSO transduction or by a dynamic programming algorithm that works on the input suboptimal decomposition. This part of the proof essentially corresponds to the machinery of typical sequences of Bodlaender and Kloks. However, we find the approach via alternation more intuitive and combinatorially less complicated, and we hope that it will find applications for computing other width measures. In fact, a similar approach has very recently been used by Giannopoulou et al. [10] in a much simpler setting of cutwidth to give a new fixed-parameter algorithm for this graph parameter.

Next, we derive a corollary of the Dealternation Lemma called the *Conflict Lemma*, which directly prepares us to construct the MSO transduction for the Bodlaender-Kloks problem. The Conflict Lemma is stated in purely combinatorial terms, but intuitively it shows that some optimum-width tree decomposition of the graph can be interpreted in the given suboptimum-width tree decomposition using subtrees that cross each other in a restricted fashion, guessable in MSO. Finally, we formalize the intuition given by the Conflict Lemma in MSO, thus constructing the MSO transduction promised in our main result.

## 2 Preliminaries and statement of the main result

**Trees, forests and tree decompositions.** Throughout this paper all graphs are undirected, unless explicitly stated. A *forest* (which is sometimes called a *rooted forest* in other contexts) is defined to be an acyclic graph, where every connected component has one designated node called the *root*. This naturally imposes parent–child and ancestor–descendant relations in a (rooted) forest. We use the usual tree terminology: root, leaf, child, parent, descendant and ancestor. We assume that every node is its own descendant, to exclude staying in the same node we use the name *strict descendant*. Likewise for ancestors. For forests we often use the name *node* instead of vertex. A tree is the special case of a forest that is connected and thus



has one root. Two nodes in a forest are called siblings if they have a common parent, or if they are both roots. Note that there is no order on siblings, unlike some models of unranked trees and forests where siblings are ordered from left to right.

A *tree decomposition* of a graph  $G$  is a pair  $t = (F, \beta)$ , where  $F$  is a rooted forest and  $\beta$  is a function that associates *bags* to the nodes of  $F$ . A bag is a nonempty subset of vertices of  $G$ . We require the following two properties:

- (T1) whenever  $uv$  is an edge of  $G$ , then there exists a node of  $F$  whose bag contains both  $u$  and  $v$ ; and
- (T2) for every vertex  $u$  of  $G$ , the set of nodes of  $F$  whose bags contain  $u$  is nonempty and induces a connected subtree in  $F$ .

The *width* of a tree decomposition is its maximum bag size minus 1, and the *treewidth* of a graph is the minimum width of its tree decomposition. An *optimum-width* tree decomposition is one whose width is equal to the treewidth of the underlying graph. Note that throughout this paper all tree decompositions will be rooted forests. This slightly diverges from the literature where usually the shape of a tree decomposition is an unrooted tree.

For a tree decomposition  $t = (F, \beta)$  of a graph  $G$ , and each node  $x$  of  $F$ , we define the following vertex sets:

- The *adhesion* of  $x$ , denoted  $\sigma(x)$ , is equal to  $\beta(x) \cap \beta(x')$ , where  $x'$  is the parent of  $x$  in  $F$ . If  $x$  is a root of  $F$ , we define its adhesion to be empty.
- The *margin* of  $x$ , denoted  $\mu(x)$ , is equal to  $\beta(x) \setminus \sigma(x)$ .
- The *component* of  $x$ , denoted  $\alpha(x)$ , is the union of the margins of all the descendants of  $x$  (including  $x$  itself). Equivalently, it is the union of the bags of all the descendants of  $x$ , minus the adhesion of  $x$ .

Whenever the tree decomposition  $t$  is not clear from the context, we specify it in the subscript, i.e., we use operators  $\beta_t(\cdot)$ ,  $\sigma_t(\cdot)$ ,  $\mu_t(\cdot)$ , and  $\alpha_t(\cdot)$ .

Observe that, by property (T2) of a tree decomposition, for every vertex of  $G$  there is a unique node whose bag contains  $u$ , but the bag of its parent (if exists) does not contain  $u$ . In other words, there is a unique node whose margin contains  $u$ . Consequently, the margins of the nodes of a tree decomposition form a partition of the vertex set of the underlying graph.

**Relational structures and MSO.** Define a *vocabulary* to be a finite set of *relation names*, each with associated arity that is a nonnegative integer. A *relational structure* over the vocabulary  $\Sigma$  consists of a set called the *universe*, and for each relation name in the vocabulary, an associated relation of the same arity over the universe. To describe properties of relational structures, we use logics, mainly *monadic second-order logic* (MSO for short). This logic allows quantification both over single elements of the universe and also over subsets of the universe. For a precise definition of MSO, see [7].

We use MSO to describe properties of graphs and tree decompositions. To do this, we need to model graphs and tree decompositions as relational structures. A graph is viewed as a relational structure, where the universe is a disjoint union of the vertex set and the edge set of a graph. There is a single binary incidence relation, which selects a pair  $(v, e)$  whenever  $v$  is a vertex and  $e$  is an incident edge. The edges can be recovered as those elements of the universe which appear on the second coordinate of the incidence relation; the vertices can be recovered as the rest of the universe. For a tree decomposition of a graph  $G$ , the universe of the corresponding structure consists of the disjoint union of: the vertex set of  $G$ , the edge set of  $G$ , and the node set of the tree decomposition. There is the incidence relation between vertices and edges, as for graphs, a binary descendant relation over the nodes of the tree decomposition, and a binary bag relation which selects pairs  $(v, x)$  such that  $x$  is

a node of the tree decomposition whose bag contains vertex  $v$  of the graph. The nodes of the decomposition can be recovered as those which are their own descendants, since we assume that the descendant relation is reflexive. Note that thus, the representation of a tree decomposition as a relational structure contains the underlying graph as a substructure.

**MSO transductions.** Suppose that  $\Sigma$  and  $\Gamma$  are vocabularies. Define a *transduction* with input vocabulary  $\Sigma$  and output vocabulary  $\Gamma$  to be a set of pairs

(input structure over  $\Sigma$ , output structure over  $\Gamma$ )

which is invariant under isomorphism of relational structures. When talking about transductions on graphs or tree decompositions, we use the representations described in the previous paragraph. Note that a transduction is a relation and not necessarily a function, thus it can have many different possible outputs for the same input. A transduction is called *deterministic* if it is a partial function (up to isomorphism). For example, the subgraph relation is a transduction from graphs to graphs, but it is not deterministic since a graph can have many subgraphs. On the other hand, the transformation that inputs a tree decomposition and outputs its underlying graph is a deterministic transduction.

This paper uses MSO transductions, as defined in the book of Courcelle and Engelfriet [7], which are a special case of transductions that can be defined using the logic MSO. The precise definition is in Section 5, but the main idea is that an MSO transduction is a finite composition of transductions of the following types: copy the input a fixed number of times, nondeterministically color the universe of the input, and add new predicates to the vocabulary with interpretations given by MSO formulas over the input vocabulary. We refer to Courcelle and Engelfriet [7] for a broader discussion of the role of MSO transduction in the theory of formal languages for graphs.

**The main result.** We now state the main contribution of this paper, which is an MSO version of the algorithm of Bodlaender and Kloks.

► **Theorem 2.** *For every  $k \in \{0, 1, 2, \dots\}$  there is an MSO transduction from tree decompositions to tree decompositions such that for every input tree decomposition  $t$ :*

- *if  $t$  has width at most  $k$ , then there is at least one output; and*
- *every output is an optimum-width tree decomposition of the underlying graph of  $t$ .*

We remark that the transduction of Theorem 2 is not deterministic, i.e. it might have several outputs on the same input. Using Theorem 2, we prove that an MSO transduction can compute an optimum-width tree decomposition given only the graph.

► **Corollary 3.** *For every  $k \in \{0, 1, 2, \dots\}$  there is an MSO transduction from graphs to tree decompositions such that for every input graph  $G$ :*

- *if  $G$  has treewidth at most  $k$ , then there is at least one output; and*
- *every output is a tree decomposition of  $G$  of optimum width.*

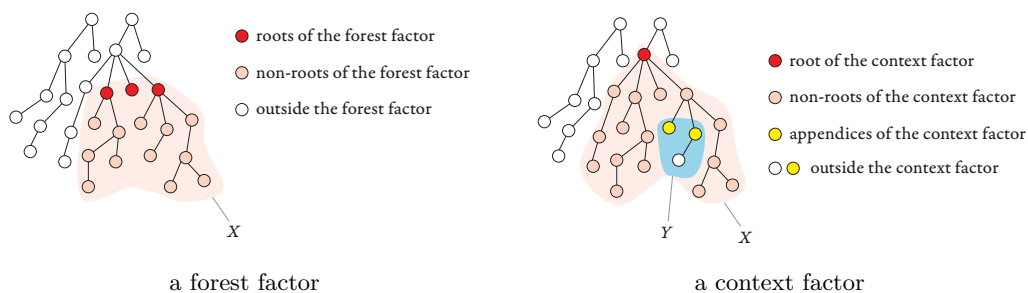
**Proof.** Theorem 2.4 of [6] says that for every  $k \in \{0, 1, 2, \dots\}$  there is an MSO transduction with exactly the properties stated in the statement, except that when the input has treewidth  $k$ , then the output tree decompositions have width at most  $f(k)$ , for some function  $f: \mathbb{N} \rightarrow \mathbb{N}$ . By composing this transduction with the transduction given by Theorem 2, applied to  $f(k)$ , we obtain the claim. ◀

**Structure of the paper.** The rest of this paper is devoted to the proof of Theorem 2. In Section 3 we formulate the Dealternation Lemma. Intuitively, this result says that any optimum-width tree decomposition  $s$  can be adjusted without increasing the width so that it behaves nicely with respect to the input suboptimal decomposition  $t$  in the following sense: for every subtree of  $t$ , the vertices appearing in the bags of this subtree are partitioned into few “connected blocks” in  $s$ . This part holds the essence of typical sequences of Bodlaender and Kloks [4], but in the full version of the paper (see <https://arxiv.org/abs/1701.06937>) we give a self-contained proof in order to achieve stronger assertions and highlight the key combinatorial properties we use later on. Then, we prove a corollary of the Dealternation Lemma, which we call the Conflict Lemma. This result intuitively states that some optimum-width tree decomposition of the graph can be interpreted in the given suboptimum-width tree decomposition. This intuition is formalized in the last section, where we introduce formally MSO transductions and use the combinatorial property given by the Conflict Lemma to prove Theorem 2.

### 3 Dealternation

This section is devoted to the Dealternation Lemma, which intuitively says that for a tree decomposition  $t$  of bounded, though possibly suboptimal width, there always exists an optimum-width decomposition in which every subtree of  $t$  is broken into small number of “pieces”. We begin by defining factors, which is our notion of “pieces” of a tree decomposition.

**Factors and factorizations.** Intuitively, a factor is a set of nodes in a forest that respects the tree structure. We define three kinds of factors: tree factors, forest factors, and context factors. A *tree factor* in a forest is a set of nodes obtained by taking all (not necessarily strict) descendants of some node, which is called the *root* of the tree factor. Define a *forest factor* to be a nonempty union of tree factors whose roots are siblings. These roots are called the *roots* of the forest factor. In particular, a tree factor is also a forest factor, with one root.



A *context factor* is the difference  $X - Y$  for a tree factor  $X$  and a forest factor  $Y$ , where the root of  $X$  is a strict ancestor of every root of  $Y$ . For a context factor  $X - Y$ , its root is defined to be the root of  $X$ , while the roots of  $Y$  are called the *appendices*. Note that a context factor always contains a unique node that is the parent of all its appendices.

Forest factors and context factors will be jointly called *factors*. The following lemma can be proved by a straightforward case study, and hence we leave its proof to the reader.

► **Lemma 4.** *The union of two intersecting factors in the same forest is also a factor.*

## 15:6 Optimizing Tree Decompositions in MSO

For a subset  $U$  of nodes of a forest, a  $U$ -factor is a factor that is entirely contained in  $U$ . A *factorization* of  $U$  is a partition of  $U$  into  $U$ -factors. A  $U$ -factor is *maximal* if no other  $U$ -factor contains it as a strict subset.

► **Lemma 5.** *For every subset of nodes  $U$  in a forest, the maximal  $U$ -factors form a factorization of  $U$ .*

**Proof.** Every node of  $U$  is contained in some factor, e.g., a singleton factor (which has forest or context type depending on whether the node is a leaf or not). Thus, every node of  $U$  is also contained in some maximal  $U$ -factor. On the other hand, two different maximal  $U$ -factors must be disjoint, since otherwise by Lemma 4, their union would also be a  $U$ -factor, contradicting maximality. ◀

The set of all maximal  $U$ -factors will be called the *maximal factorization* of  $U$ , and will be denoted by  $\text{fact}(U)$ . We specify the forest in the subscript whenever it is not clear from the context. Lemma 5 asserts that  $\text{fact}(U)$  is indeed a factorization of  $U$ . Note that the maximal factorization of  $U$  is the coarsest in the following sense: in every factorization of  $U$ , each of its factors is contained in some factor of  $\text{fact}(U)$ . In particular, the maximal factorization has the smallest number of factors among all factorizations of  $U$ .

In the sequel, we will need the following simple result about relation between the maximal factorizations of a set and of its complement. Its proof is a part of the proof of the Dealternation Lemma, and can be found in the full version of the paper.

► **Lemma 6.** *Suppose  $(U, W)$  is a partition of the node set of a rooted forest  $F$ , and let  $k$  be the number of factors in the maximal factorization of  $W$ . Then the maximal factorization of  $U$  has at most  $k + 1$  forest factors and at most  $2k - 1$  context factors.*

**Separation forests.** The general definition of a tree decomposition is flexible and allows for multiple combinatorial adjustments. Here, we will rely on a normalized form that we call *separation forests*, which are essentially tree decompositions where all the margins have size exactly 1. The definition of treewidth via separation forests resembles the definition of pathwidth via the so-called *vertex separation number* [14].

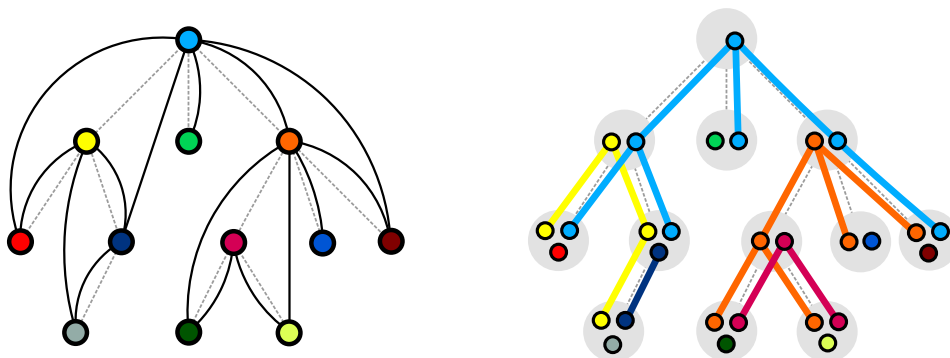
► **Definition 7.** Suppose  $G$  is a graph. A *separation forest* of  $G$  is a rooted forest  $F$  on the same vertex set as  $G$  such that  $G$  is contained in the ancestor-descendant closure of  $F$ ; that is, whenever  $uv$  is an edge of  $G$ , then  $u$  is an ancestor of  $v$  in  $F$  or vice versa.

Separation forests are used to define the graph parameter *treedepth*, which is equal to the minimum depth of a separation forest of a graph. To define treewidth, we need to take a different measure than just the depth, as explained next.

Suppose  $F$  is a separation forest of  $G$ . Endow  $F$  with the following bag function  $\beta(\cdot)$ . For any vertex  $u$  of  $G$ , assign to  $u$  the bag  $\beta(u)$  consisting of  $u$  and all the ancestors of  $u$  in  $F$  that have a neighbor among the descendants of  $u$  in  $F$ . The following claim follows by verifying the definition of a tree decomposition; we leave the easy proof to the reader.

► **Claim 8.** *If  $F$  is a separation forest of  $G$  and  $\beta(\cdot)$  is defined as above, then  $(F, \beta)$  is a tree decomposition of  $G$ . Moreover, for every vertex  $u$  of  $G$ , the margin of  $u$  in  $(F, \beta)$  is  $\{u\}$ .*

The tree decomposition  $(F, \beta)$  defined above is said to be *induced* by the separation forest  $F$ . Observe that if  $t = (F, \beta)$  is induced by  $F$ , then for any vertex  $u$ , the component of  $u$  in  $t$  consists of all the descendants of  $u$  in  $F$ .



■ **Figure 1** Construction of the induced tree decomposition from a separation forest. The graph edges are depicted in black, the child-parent relation of the forest is depicted as dashed grey lines.

One can reformulate the construction given above as follows. First, put every vertex  $u$  into its bag  $\beta(u)$ . Then, examine every neighbor  $v$  of  $u$ , and if  $v$  is a descendant of  $u$  in  $F$ , then add  $u$  to every bag on the path from  $v$  to  $u$  in  $F$ . Thus, every vertex  $u$  is “smeared” onto a subtree of  $F$ , where  $u$  is the root of this subtree and its leaves correspond to those neighbors of  $u$  that are also its descendants in  $F$ . This construction is depicted in Figure 1.

The *width* of a separation forest is simply the width of the tree decomposition induced by it. Consequently, the width of a separation forest is never smaller than the treewidth of a graph. The next result shows that in fact there is always a separation forest of optimum width. The proof follows by a simple surgery on an optimum-width tree decomposition, and can be found in the full version of the paper.

► **Lemma 9.** *For every graph  $G$  there exists a separation forest of  $G$  whose width is equal to the treewidth of  $G$ .*

**Dealternation Lemma.** We are finally ready to state the Dealternation Lemma.

► **Lemma 10 (Dealternation Lemma).** *There exist functions  $f(k) \in \mathcal{O}(k^2)$  and  $g(k) \in \mathcal{O}(k^3)$  such that the following holds. Suppose that  $t$  is a tree decomposition of a graph  $G$  of width  $k$ . Then there exists an optimum-width separation forest  $F$  of  $G$  such that:*

- (D1) *for every node  $x$  of  $t$ , the maximal factorization  $\text{fact}_F(\alpha_t(x))$  has at most  $f(k)$  factors;*
- (D2) *for every node  $x$  of  $t$ , there are at most  $g(k)$  children of  $x$  in the set*

$$\{y: y \text{ is a node of } t \text{ with at least one context factor in } \text{fact}_F(\alpha_t(y))\}.$$

Note that in the statement of the Dealternation Lemma, the vertex set of  $G$  is at the same time the node set of the forest  $F$ . Thus,  $\text{fact}_F(\alpha_t(x))$  denotes the maximal factorization of  $\alpha_t(x)$ , treated as a subset of nodes of  $F$ .

The proof of the Dealternation Lemma uses essentially the same core ideas as the correctness proof of the algorithm of Bodlaender and Kloks [4]. We include our proof in the full version of the paper for several reasons. First, unlike in [4], in our setting we cannot assume that  $t$  has binary branching, as is the case in [4]. In fact, condition (D2) is superfluous when  $t$  has binary branching. Second, our formulation of the Dealternation Lemma highlights the key combinatorial property, which is expressed as the existence of a single separation forest  $F$  that behaves nicely with respect to the input decomposition  $t$ . This property is somehow implicit [4], where the existence of nicely-behaved optimum-width tree decompositions is argued along performing dynamic programming. For this reason, we find the new formulation more explanatory and potentially interesting on its own.

#### 4 Using the Dealternation Lemma

In this section we use the Dealternation Lemma to show that an optimum-width separation forest of a graph can be interpreted in a suboptimum-width tree decomposition. For this, we need to develop a better understanding of the combinatorial insight provided by the Dealternation Lemma, which is expressed via an auxiliary graph, called the *conflict graph*.

Suppose  $G$  is a graph,  $t$  is a tree decomposition of  $G$  of width  $k$ , and  $F$  is a separation forest of  $G$ . Let  $\phi$  be the mapping that sends each vertex  $u$  of  $G$  to the unique node of  $t$  that contains  $u$  in its margin. For a vertex  $u$  of  $G$ , we define the *stain* of  $u$ , denoted  $S_u$ , which is a subgraph of the underlying forest of  $t$ , as follows. For every child  $v$  of  $u$  in  $F$ , find the unique path in  $t$  between  $\phi(u)$  and  $\phi(v)$ . Then stain  $S_u$  consists of the node  $\phi(u)$  and the union of these paths. Note that if  $u$  is a leaf of  $F$ , then the stain  $S_u$  consists only of the node  $\phi(u)$ . Define the *conflict graph*  $H(t, F)$  as follows. The vertices of  $H(t, F)$  are the vertices of  $G$ , and vertices  $u$  and  $v$  are adjacent in  $H(t, F)$  if and only their stains  $S_u$  and  $S_v$  have a node in common. The main result of this section can be formulated as follows.

► **Lemma 11** (Conflict Lemma). *There is a function  $h(k) \in \mathcal{O}(k^5)$  such that if  $t$  and  $F$  are as in the Dealternation Lemma, then their conflict graph  $H(t, F)$  admits a proper coloring with  $h(k)$  colors.*

Recall here that a proper coloring of a graph is a coloring of its vertex set such that no two adjacent vertices receive the same color. The rest of this section is devoted to the proof of the Conflict Lemma. From now on, we assume that  $G, t, F$  are as in the Dealternation Lemma, and we denote  $H = H(t, F)$ .

Observe that the conflict graph  $H$  is an intersection graph of a family of subtrees of a forest. It is well-known (see, e.g., [11]) that this property precisely characterizes the class of chordal graphs (graphs with no induced cycle of length larger than 3), so  $H$  is chordal. Chordal graphs are known to be perfect (again see, e.g., [11]), hence the chromatic number of a chordal graph (the minimum number of colors needed in a proper coloring) is equal to the size of the largest clique in it. On the other hand, subtrees of a forest are known to satisfy the so-called Helly property: whenever  $\mathcal{F}$  is some family of subtrees such that the subtrees in  $\mathcal{F}$  pairwise intersect, then in fact there is a node of the forest that belongs to all the subtrees in  $\mathcal{F}$ . This means that the largest clique in an intersection graph of a family of subtrees of a forest can be obtained by taking all the subtrees that contain some fixed node. Therefore, to prove the Conflict Lemma it is sufficient to prove the following claim.

► **Claim 12.** *There exists a function  $h(k) \in \mathcal{O}(k^5)$  such that every node of  $t$  belongs to at most  $h(k)$  of the stains  $\{S_u : u \in V(G)\}$ .*

In the remainder of this section we prove Claim 12. Fix any node  $x$  of  $t$ , and let  $y_1, y_2, \dots, y_p$  be its children in  $t$ . Consider the following partition of the vertex set of  $G$ :

$$\Pi = (\alpha_t(y_1), \alpha_t(y_2), \dots, \alpha_t(y_p), \mu_t(x), V(G) \setminus \alpha_t(x))$$

Define a factorization  $\Phi$  of the whole node set of  $F$  as follows: for each set  $X$  from the partition  $\Pi$ , take its maximal factorization  $\text{fact}_F(X)$ , and define  $\Phi$  to be the union of these maximal factorizations. Thus,  $\Phi$  is a factorization that refines the partition  $\Pi$ . Since the number of children  $y_i$  is unbounded, we cannot expect that  $\Phi$  has a small number of factors, but at least it has a small number of context factors.

► **Claim 13.** *Factorization  $\Phi$  contains at most  $g(k) \cdot f(k) + 2f(k) + k$  context factors, where  $f$  and  $g$  are as in the Dealternation Lemma.*

**Proof.** By the Dealternation Lemma, the maximal factorization in  $F$  of each of the sets  $\alpha_t(y_1), \dots, \alpha_t(y_p), \alpha_t(x)$  has at most  $f(k)$  factors. Moreover, only at most  $g(k)$  of the sets  $\alpha_t(y_1), \dots, \alpha_t(y_p)$  can have a context factor in their maximal factorizations. Hence, the maximal factorizations of sets  $\alpha_t(y_1), \dots, \alpha_t(y_p)$  introduce at most  $g(k) \cdot f(k)$  context factors to the factorization  $\Pi$ . Since the maximal factorization of  $\alpha_t(x)$  has at most  $f(k)$  factors as well, by Lemma 6 we deduce that the maximal factorization of  $V(G) \setminus \alpha_t(x)$  has at most  $2f(k) - 1$  context factors. Finally, the cardinality of  $\mu_t(x)$  is at most  $k + 1$ , so in particular its maximal factorization has at most  $k + 1$  factors in total. Summing up all these upper bounds, we conclude that  $\Phi$  has at most  $g(k) \cdot f(k) + 2f(k) + k$  context factors. ◀

With Claim 13 in hand, we complete now the proof of Claim 12. Take any vertex  $u$  such that  $x$  belongs to the stain  $S_u$ . This means that either

- (i)  $u$  belongs to the margin of  $x$ , or
- (ii)  $u$  does not belong to the margin of  $x$ , but  $u$  has a child  $v$  in  $F$  such that the unique path in  $t$  between  $\phi(u)$  and  $\phi(v)$  passes through  $x$ .

The number of vertices  $u$  satisfying 1 is bounded by the size of the margin of  $x$ , which is at most  $k + 1$ , hence we focus on vertices  $u$  that satisfy 2. Observe that condition 2 in particular means that  $u$  and  $v$  belong to different parts of partition  $\Pi$ , so also to different factors of factorization  $\Phi$ . Since  $u$  is the parent of  $v$  in  $F$ , this means that the unique factor of  $\Phi$  that contains  $u$  must be a context factor, and  $u$  must be the parent of its appendices. Consequently, the number of vertices  $u$  satisfying 2 is upper bounded by the number of context factors in factorization  $\Phi$ , which is at most  $g(k) \cdot f(k) + 2f(k) + k$  by Claim 13. We conclude that the number of stains  $S_u$  containing  $x$  is at most

$$h(k) := g(k) \cdot f(k) + 2f(k) + 2k + 1;$$

in particular  $h(k) \in \mathcal{O}(k^5)$ . This concludes the proof of Claim 12, so also the proof of the Conflict Lemma is complete.

## 5 Constructing the transduction

We now use the understanding gathered in the previous sections to give an MSO transduction that takes a tree decomposition of a graph of suboptimum width, and produces an optimum-width tree decomposition. First, we need to precisely define MSO transductions.

**MSO transductions.** Formally, an MSO transduction is any transduction that can be obtained by composing a finite number of transductions of the following kinds. Note that kind 1 is a partial function, kinds 2, 3, 4 are functions, and kind 5 is a relation.

1. **Filtering.** For every MSO sentence  $\varphi$  over the input vocabulary there is transduction that filters out structures where  $\varphi$  is satisfied. Formally, the transduction is the partial identity whose domain consists of the structures that satisfy the sentence. The input and output vocabularies are the same.
2. **Universe restriction.** For every MSO formula  $\varphi(x)$  over the input vocabulary with one free first-order variable there is a transduction, which restricts the universe to those elements that satisfy  $\varphi$ . The input and output vocabularies are the same, the interpretation of each relation in the output structure is defined as the restriction of its interpretation in the input structure to tuples of elements that remain in the universe.
3. **MSO interpretation.** This kind of transduction changes the vocabulary of the structure while keeping the universe intact. For every relation name  $R$  of the output vocabulary,

there is an MSO formula  $\varphi_R(x_1, \dots, x_k)$  over the input vocabulary which has as many free first-order variables as the arity of  $R$ . The output structure is obtained from the input structure by keeping the same universe, and interpreting each relation  $R$  of the output vocabulary as the set of those tuples  $(x_1, \dots, x_k)$  that satisfy  $\varphi_R$ .

4. **Copying.** For  $k \in \{1, 2, \dots\}$ , define  $k$ -copying to be the transduction which inputs a structure and outputs a structure consisting of  $k$  disjoint copies of the input. Precisely, the output universe consists of  $k$  copies of the input universe. The output vocabulary is the input vocabulary enriched with a binary predicate `copy` that selects copies of the same element, and unary predicates `layer1`, `layer2`,  $\dots$ , `layerk` which select elements belonging to the first, second, etc. copies of the universe. In the output structure, a relation name  $R$  of the input vocabulary is interpreted as the set of all those tuples over the output structure, where the original elements of the copies were in relation  $R$  in the input structure.
5. **Coloring.** We add a new unary predicate to the input structure. Precisely, the universe as well as the interpretations of all relation names of the input vocabulary stay intact, but the output vocabulary has one more unary predicate. For every possible interpretation of this unary predicate, there is a different output with this interpretation implemented.

We remark that the above definition is easily equivalent to the one used in [6], where filtering, universe restriction, and MSO interpretation are merged into one kind of a transduction.

**Proving the main result.** We are finally ready to prove our main result, Theorem 2. The proof is broken down into several steps. The first, main step shows that an MSO transduction can output optimum-width separation forests. Here, a separation forest of a graph  $G$  is encoded by enriching the relational structure encoding  $G$  with a single binary relation interpreted as the child relation of  $F$ . Note that the definition of a separation forest is MSO-expressible: there is an MSO sentence that checks whether the additional relation indeed encodes a separation forest of the graph.

► **Lemma 14.** *For every  $k \in \{0, 1, 2, \dots\}$ , there is an MSO transduction from tree decompositions to separation forests such that for every input tree decomposition  $t$ :*

- *every output is a separation forest of the underlying graph of  $t$ ; and*
- *if  $t$  has width at most  $k$ , then there is at least one output that is a separation forest of optimum width.*

**Proof.** Observe that the verification whether the width of  $t$  is at most  $k$  can be expressed by an MSO sentence, so we can first use filtering to filter out any input tree decomposition  $t$  whose width is larger than  $k$ ; for such decompositions, the transduction produces no output. Let  $G$  be the underlying graph of  $t$ , and let  $\phi$  be the mapping that sends each vertex  $u$  of  $G$  to the unique node of  $t$  whose margin contains  $u$ . By the Conflict Lemma, there exists some separation forest  $F$  of  $G$  of optimum width such that the conflict graph  $H(t, F)$  admits some proper coloring  $\lambda$  with  $h(k)$  colors. The constructed MSO transduction attempts at guessing and interpreting  $F$  as follows.

First, using coloring and filtering, we guess the coloring  $\lambda$ , represented as a partition of the vertex set of  $G$ . Then, again using coloring and filtering, for every vertex  $u$  of  $G$  we guess whether  $u$  is a root of  $F$ , and if not, then we guess the color under  $\lambda$  of the parent of  $u$  in  $F$ .

Next, for every color  $c$  used in  $\lambda$ , we guess the forest

$$M_c := \bigcup_{u \in \lambda^{-1}(c)} S_u,$$

where  $S_u$  is the stain of  $u$  in  $t$ , defined as in Section 4 for the separation forest  $F$ . Note that the stains  $\{S_u : u \in \lambda^{-1}(c)\}$  are pairwise disjoint, because  $\lambda$  is a proper coloring of



the conflict graph  $H(t, F)$ . Thus, the connected components of  $M_c$  are exactly these stains. Observe also that  $M_c$  is a subgraph of the decomposition  $t$ , so we can emulate guessing  $M_c$  in an MSO transduction working over  $t$  by guessing the subset of those nodes of  $t$ , for which the edge of  $t$  connecting the node and its parent belongs to  $M_c$ .

Having done all these guesses, we can interpret the child relation of  $F$  using an MSO predicate as follows. Fix a pair of vertices  $u$  and  $v$ , and let  $c$  be the guessed color of  $u$  under  $\lambda$ . Then one can readily check that  $u$  is the parent of  $v$  in  $F$  if and only if the following conditions are satisfied:

- we have guessed that  $v$  is not a root of  $F$ ,
- we have guessed that the color of the parent of  $v$  in  $F$  is  $c$ , and
- $u$  is the unique vertex of color  $c$  such that  $\phi(u)$  belongs to the same connected component of  $M_c$  as  $\phi(v)$ .

It can be easily seen that these conditions can be expressed by an MSO formula with two free variables  $u$  and  $v$ .

Finally, we filter out all the wrong guesses by verifying, using an MSO sentence, whether the interpreted child relation on the vertices of  $G$  indeed forms a rooted forest, and whether this forest is a separation forest of  $G$ . Obviously, the separation forest  $F$  was obtained for at least one of the guesses, and survives this filtering. At the end, we remove the nodes of decomposition  $t$  from the structure using universe restriction. ◀

Next, we need to construct the induced tree decomposition out of a separation forest.

► **Lemma 15.** *There is an MSO transduction from separation forests to tree decompositions that on each input separation forest has exactly one output, which is the tree decomposition induced by the input.*

**Proof.** We copy the vertex set of the graph two times, and declare the second copies to be the nodes of the constructed tree decomposition. Using the child relation of the input separation forest, we can interpret in MSO the descendant relation in the forest of the decomposition. Finally, the bag relation in the induced tree decomposition, as defined in Section 3, can be easily interpreted using an MSO formula. ◀

Finally, so far the transduction can output tree decompositions of suboptimal width, which should be filtered out. For this, we need the following MSO-expressible predicate.

► **Lemma 16.** *For every  $k \in \{0, 1, 2, \dots\}$ , there is an MSO-sentence over tree decompositions that holds if and only if the given tree decomposition has width at most  $k$  and its width is optimum for the underlying graph.*

**Proof.** Let  $t$  be the given tree decomposition of a graph  $G$ . Obviously, we can verify using an MSO sentence whether the width of  $t$  is at most  $k$ . To check that the width of  $t$  is optimum, we could use the fact that graphs of treewidth  $k$  are characterized by a finite list of forbidden minors, but we choose to apply the following different strategy. Let  $R_k$  be the MSO transduction that is the composition of the transductions of Lemmas 14 (for parameter  $k$ ) and 15. Provided the input tree decomposition  $t$  has width at most  $k$ , transduction  $R_k$  outputs some set of tree decompositions of  $G$  among which one has optimum width. Hence,  $t$  has optimum width if and only if the output  $R_k(t)$  does not contain any tree decomposition of width smaller than  $t$ .

The Backwards Translation Theorem for MSO transductions [7] states that whenever  $T$  is an MSO transduction and  $\psi$  is an MSO sentence over the output vocabulary, then the set of structures on which  $T$  outputs at least one structure satisfying  $\psi$ , is MSO-definable over

the input vocabulary. Hence, for every  $p < k$ , there exists an MSO sentence  $\varphi_p$  that verifies whether  $R_k(t)$  outputs at least one tree decomposition of width at most  $p$ . Therefore, we can check whether  $t$  has optimum width by making a disjunction over all  $\ell$  with  $0 \leq \ell \leq k$  of the sentences stating that  $t$  has width exactly  $\ell$  and  $R_k(t)$  does not output any tree decomposition of width less than  $\ell$ . ◀

Theorem 2 now follows by composing the MSO transductions given by Lemmas 14 and 15, and at the end applying filtering using the predicate given by Lemma 16.

## 6 Conclusions

In this work we have constructed an MSO transduction that, given a constant-width tree decomposition of a graph, computes a tree decomposition of this graph of optimum width. As we have shown, this transduction can be conveniently composed with the MSO transduction given in [6] to prove that given a graph of constant treewidth, some optimum-width tree decomposition can be computed by means of an MSO transduction.

One direct application of this result is a strengthening of the main result of [6]. There, we have proved that if a class of graphs of treewidth at most  $k$  is recognizable (see [6] for omitted definitions), then it can be defined in MSO with modular counting predicates. The main technical component of this proof was Theorem 2.4, which states that for every  $k$  there is an MSO transduction from graphs to tree decompositions, which given a graph of treewidth  $k$  outputs some its tree decomposition of width bounded by  $f(k)$ , for some doubly-exponential function  $f$ . Then the proof of the main result of [6] used  $f(k)$ -recognizability, i.e., recognizability within the interface (sourced) graphs with at most  $f(k)$  interfaces (sources). By replacing the usage of Theorem 2.4 of [6] with Corollary 3 of this paper, we deduce that only  $k$ -recognizability of a class of graphs of treewidth at most  $k$  is sufficient to prove that it can be defined in MSO with modular counting predicates. However, this strengthening was already known: Courcelle and Lagergren [8] proved that if a class of graphs of treewidth at most  $k$  is  $k$ -recognizable, then it is also  $k'$ -recognizable for all  $k' \geq k$ . In fact, the proof technique of Courcelle and Lagergren essentially uses the same technique as Bodlaender and Kloks [4] and as we do in this work; the main technical component of [8] can be interpreted as a variant of our Local Dealternation Lemma (see the full version of the paper).

Finally, we see potential algorithmic applications of our main result. Namely, it seems that the existing results, in particular the literature on constructing answers to MSO queries on trees [1, 9, 13], are likely to imply the following algorithmic statement. Suppose  $R$  is an MSO transduction whose domain are rooted forests labelled by a finite alphabet. Then, given a forest  $t$ , one can compute in time  $f(k) \cdot (n + m)$  any member of the output of  $R$  on  $t$ , or conclude that this output is empty. Here,  $n$  is the size of  $t$ ,  $m$  is the size of the output structure (or 0 if transduction  $R$  applied to  $t$  yields no output),  $k$  is the size of the description of  $R$ , and  $f$  is some function. Assuming such an algorithmic statement, the algorithmic result of Bodlaender and Kloks [4] (without a specified dependency on  $k$  of the running time) would basically follow from applying it to the MSO transduction constructed in this paper. However, such a tool could be of more general use. It would essentially reduce designing constructive dynamic programming algorithms on tree decompositions, which most often is a tedious and complicated task, to describing the corresponding transformations using MSO transductions, similarly as Courcelle's theorem reduces designing dynamic programming algorithm for decision problems to expressing them in MSO. We will explore these algorithmic applications in the journal version of our work.

**Acknowledgements.** The authors would like to thank Bruno Courcelle for pointing out connections with his work with Jens Lagergren [8].

---

**References**

---

- 1 Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *CSL 2006*, volume 4207 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2006.
- 2 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.
- 3 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshтанov, and Michał Pilipczuk. A  $c^kn$  5-approximation algorithm for treewidth. *SIAM J. Comput.*, 45(2):317–378, 2016.
- 4 Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, 1996.
- 5 Hans L. Bodlaender and Dimitrios M. Thilikos. Constructive linear time algorithms for branchwidth. In *ICALP 1997*, volume 1256 of *Lecture Notes in Computer Science*, pages 627–637. Springer, 1997.
- 6 Mikołaj Bojańczyk and Michał Pilipczuk. Definability equals recognizability for graphs of bounded treewidth. In *LICS 2016*, pages 407–416. ACM, 2016.
- 7 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.
- 8 Bruno Courcelle and Jens Lagergren. Equivalent definitions of recognizability for sets of graphs of bounded tree-width. *Mathematical Structures in Computer Science*, 6(2):141–165, 1996.
- 9 Jörg Flum, Markus Frick, and Martin Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, 2002.
- 10 Archontia C. Giannopoulou, Michał Pilipczuk, Jean-Florent Raymond, Dimitrios M. Thilikos, and Marcin Wrochna. Cutwidth: obstructions and algorithmic aspects. *CoRR*, abs/1606.05975, 2016. To appear in Proc. of IPEC 2016.
- 11 Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs*, volume 57 of *Annals of Discrete Mathematics*. North-Holland Publishing Co., 2004.
- 12 Jisu Jeong, Eun Jung Kim, and Sang-il Oum. Constructive algorithm for path-width of matroids. In *SODA 2016*, pages 1695–1704. SIAM, 2016.
- 13 Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *ACM Trans. Comput. Log.*, 14(4):25, 2013.
- 14 Nancy G. Kinnersley. The vertex separation number of a graph equals its path-width. *Inf. Process. Lett.*, 42(6):345–350, 1992.
- 15 Jens Lagergren and Stefan Arnborg. Finding minimal forbidden minors using a finite congruence. In *ICALP 1991*, volume 510 of *Lecture Notes in Computer Science*, pages 532–543. Springer, 1991.
- 16 Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Cutwidth I: A linear time fixed parameter algorithm. *J. Algorithms*, 56(1):1–24, 2005.
- 17 Dimitrios M. Thilikos, Maria J. Serna, and Hans L. Bodlaender. Cutwidth II: Algorithms for partial  $w$ -trees of bounded degree. *J. Algorithms*, 56(1):25–49, 2005.



# Complexity of Token Swapping and its Variants\*

Édouard Bonnet<sup>1</sup>, Tillmann Miltzow<sup>2</sup>, and Paweł Rzażewski<sup>3</sup>

- 1 Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary  
edouard.bonnet@dauphine.fr
- 2 Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary  
t.miltzow@gmail.com
- 3 Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary; and  
Faculty of Mathematics and Information Science, Warsaw University of Technology, Warsaw, Poland  
p.rzazewski@mini.pw.edu.pl

---

## Abstract

In the TOKEN SWAPPING problem we are given a graph with a token placed on each vertex. Each token has exactly one destination vertex, and we try to move all the tokens to their destinations, using the minimum number of swaps, i.e., operations of exchanging the tokens on two adjacent vertices. As the main result of this paper, we show that TOKEN SWAPPING is  $W[1]$ -hard parameterized by the length  $k$  of a shortest sequence of swaps. In fact, we prove that, for any computable function  $f$ , it cannot be solved in time  $f(k)n^{o(k/\log k)}$  where  $n$  is the number of vertices of the input graph, unless the ETH fails. This lower bound almost matches the trivial  $n^{O(k)}$ -time algorithm.

We also consider two generalizations of the TOKEN SWAPPING, namely COLORED TOKEN SWAPPING (where the tokens have colors and tokens of the same color are indistinguishable), and SUBSET TOKEN SWAPPING (where each token has a set of possible destinations). To complement the hardness result, we prove that even the most general variant, SUBSET TOKEN SWAPPING, is FPT in nowhere-dense graph classes.

Finally, we consider the complexities of all three problems in very restricted classes of graphs: graphs of bounded treewidth and diameter, stars, cliques, and paths, trying to identify the borderlines between polynomial and NP-hard cases.

**1998 ACM Subject Classification** G.2.2 Graph Theory, F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** token swapping, parameterized complexity, NP-hardness,  $W[1]$ -hardness

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.16

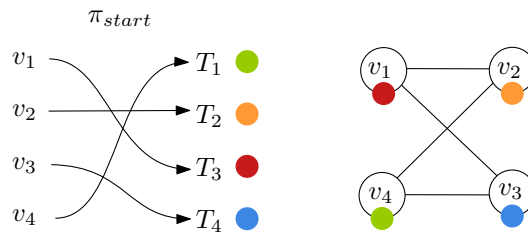
## 1 Introduction

In reconfiguration problems, one is interested in transforming a combinatorial or geometric object from one state to another, by performing a sequence of simple operations. An important example is motion planning, where we want to move an object from one configuration to another. Elementary operations are usually translations and rotations. It turns out that

---

\* Supported by the ERC grant PARAMTIGHT: “Parameterized complexity and the search for tight complexity results”, no. 280152.





■ **Figure 1** Every token placement can be uniquely described by a permutation.

motion planning can be reduced to the shortest path problem in some higher dimensional Euclidean space with obstacles [7].

Finding the shortest flip sequence between any two triangulations of a convex polygon is a major open problem in computational geometry. Interestingly it is equivalent to a myriad of other reconfiguration problems of so-called Catalan structures [4]. Examples include: binary trees, perfect matchings of points in convex position, Dyck words, monotonic lattice paths, and many more. Reconfiguring permutations under various constraints is heavily studied and usually called *sorting*.

An important class of reconfiguration problems is a big family of problems in graph theory that involves moving tokens, pebbles, cops or robbers along the edges of a given graph, in order to reach some final configuration [30, 5, 9, 14, 2, 28, 20, 8, 12]. In this paper, we study one of them.

The TOKEN SWAPPING problem, introduced by Akers and Krishnamurthy [1], and stated more recently by Yamanaka *et al.* [31], fits nicely into this long history of reconfiguration problems and can be regarded as a sorting problem with special constraints.

The problem is defined as follows, see also Figure 1. We are given an undirected connected graph with  $n$  vertices  $v_1, \dots, v_n$ , a set of tokens  $T = \{t_1, \dots, t_n\}$  and two permutations  $\pi_{\text{start}}$  and  $\pi_{\text{target}}$ . These permutations are called start permutation and target permutation. Initially vertex  $v_i$  holds token  $t_{\pi_{\text{start}}(i)}$ . In one step, we are allowed to *swap* tokens on a pair of adjacent vertices, that is, if  $v$  and  $w$  are adjacent,  $v$  holds the token  $s$ , and  $w$  holds the token  $t$ , then the swap between  $v$  and  $w$  results in the configuration where  $v$  holds  $t$ ,  $w$  holds  $s$ , and all the other tokens stay in place. The TOKEN SWAPPING problem asks if the target configuration can be reached in at most  $k$  swaps. Thus, a solution for the TOKEN SWAPPING problem is a sequence of edges, where the swaps take place. The solution is optimal if its length is shortest possible. To see the correspondence to sorting note that every placement of tokens can be regarded as a permutation and the target permutation can be regarded as the sorted state.

It has been observed, for example in [1, 31], that every instance of TOKEN SWAPPING has a solution, and its length is  $O(n^2)$ . Moreover,  $\Omega(n^2)$  swaps are sometimes necessary. It is interesting to note that some special cases of TOKEN SWAPPING have been studied in the context of sorting permutations with additional restrictions (see Knuth [21, Section 5.2.2] for paths, Pak [27] for stars, Cayley [6] for cliques, and Heath and Vergara [16] for squares of a path). Recently the problem was also solved for a special case of complete split graphs (see Yasui *et al.* [33]). It is also worth mentioning that a very closely related concept of sorting permutations using cost-constrained transitions was considered by Farnoud *et al.* [11], and Farnoud and Milenkovic [10].

The complexity of the TOKEN SWAPPING problem was investigated by Miltzow *et al.* [25]. They show that the problem is NP-complete and APX-complete. Moreover, they show that any algorithm solving the TOKEN SWAPPING problem in time  $2^{o(n)}$  would refute the Exponential Time Hypothesis (ETH) [17]. The results of Miltzow *et al.* [25] carry over also to

some generalization of the TOKEN SWAPPING problem, called COLORED TOKEN SWAPPING, first introduced by Yamanaka *et al.* [32]. In this problem, vertices and tokens are partitioned into color classes. For each color  $c$ , the number of tokens colored  $c$  equals the number of vertices colored  $c$ . The goal is to reach, with the minimum number of swaps, a configuration in which each vertex contains a token of its own color. TOKEN SWAPPING corresponds to the special case where each color class comprises exactly one token and one vertex. NP-hardness of COLORED TOKEN SWAPPING was first shown by Yamanaka *et al.* [32], even in the case where only 3 colors exist.

We introduce the SUBSET TOKEN SWAPPING problem, which is an even further generalization of TOKEN SWAPPING. Here a function  $D : T \rightarrow 2^V$  specifies the set  $D(t_i)$  of possible destinations  $D(t_i)$  for the token  $t_i$ . Observe that SUBSET TOKEN SWAPPING also generalizes COLORED TOKEN SWAPPING. It might happen that there is no satisfying swapping sequence at all to this new problem. Though, this can be checked in polynomial time by deciding if there is a perfect matching in the bipartite token-destination graph. Thus we shall always assume that we have a satisfiable instance.

In this paper we continue and extend the work of Miltzow *et al.* [25]. They presented a very simple algorithm which solves the instance of the TOKEN SWAPPING problem in  $n^{O(k)}$  time and space, where  $k$  denotes the number of allowed swaps. In Section 3 we show that this algorithm can be easily generalized to COLORED TOKEN SWAPPING and SUBSET TOKEN SWAPPING problems. One of the main bottlenecks for exponential-time algorithms is not time, but space consumption. Thus we present a slightly slower exact algorithm, using only polynomial space (in fact, only slightly super-linear).

The existence of an XP algorithm (i.e., with time complexity  $O(n^{f(k)})$  for some computable function  $f$ ) for the TOKEN SWAPPING problem gives rise to the question whether the problem can be solved in FPT time (i.e.,  $f(k) \cdot n^{O(1)}$ , for some computable function  $f$ ). There is some evidence indicating that this could be possible. First, observe that an instance with more than  $2k$  misplaced tokens, is a NO-instance, as each swap moves only two tokens. Further, one can safely remove all vertices from the graph that are at distance more than  $k$  from all misplaced tokens. This preprocessing yields an equivalent instance, where every connected component has *diameter*  $O(k^2)$ . Thus if the maximum degree  $\Delta$  is bounded by  $k$ , each component has size bounded by a function of  $k$ . The connected components of  $f(k)$  size can be solved separately by exhaustively guessing (still in FPT time) the number of swaps to perform in each of them. Moreover, even the generalized SUBSET TOKEN SWAPPING problem is FPT in  $k + \Delta$  (see Proposition 6). For those reasons, one could have hoped for an FPT algorithm for general graphs. However, as the main result of this paper, we show in Section 4 that this is very unlikely.

► **Theorem 1** (Parameterized Hardness). *TOKEN SWAPPING is  $W[1]$ -hard, parameterized by the number  $k$  of allowed swaps. Moreover, assuming the ETH, for any computable function  $f$ , TOKEN SWAPPING cannot be solved in time  $f(k)(n + m)^{o(k/\log k)}$  where  $n$  and  $m$  are respectively the number of vertices and edges of the input graph.*

Observe that this lower bound shows that the simple  $n^{O(k)}$ -time algorithm is almost best possible. It is worth mentioning that the parameter for which we show hardness is in fact *number of swaps + number of initially misplaced tokens + diameter of the graph*, which matches the reasoning presented in the previous paragraph.

To show the lower bound, we introduce handy gadgets called *linkers*. They are simple and can be used to give a significantly simpler proof of the lower bounds given by Miltzow *et al.* [25].

## 16:4 Complexity of Token Swapping and its Variants

■ **Table 1** The parameterized complexity of TOKEN SWAPPING, COLORED TOKEN SWAPPING, and SUBSET TOKEN SWAPPING.

	$k + \Delta$	$k + \text{diam}$	$k$ , nowhere-dense / $k + \text{tw}$	$\text{tw} + \text{diam}$
TS	FPT ([25])	<b>W[1]-h</b> (Th 1)	FPT	<b>paraNP-c</b> (Th 4)
CTS	FPT	W[1]-h	FPT	paraNP-c
STS	<b>FPT</b> (Prop 6)	W[1]-h	<b>FPT</b> (Th 2)	paraNP-c

Since there is no FPT algorithm for the TOKEN SWAPPING problem (parameterized by the number  $k$  of swaps), unless  $\text{FPT} = \text{W}[1]$ , a natural approach is to try to restrict the input graph classes, in hope to obtain some positive results. Indeed, in Section 5 we show that FPT algorithms exist, if we restrict our input to the so-called *nowhere-dense graph classes*.

► **Theorem 2** (FPT in nowhere dense graphs). *SUBSET TOKEN SWAPPING is FPT parameterized by  $k$  on nowhere-dense graph classes.*

The notion of nowhere-dense graph classes has been introduced as a common generalization of several previously known notions of sparsity in graphs such as planar graphs, graphs with forbidden (topological) minors, graphs with (locally) bounded treewidth or graphs with bounded maximum degree. Grohe, Kreutzer, and Siebertz [15] proved that every property definable as a first-order formula  $\varphi$  is solvable in  $O(f(|\varphi|, \varepsilon) n^{1+\varepsilon})$  time on nowhere-dense classes of graphs, for every  $\varepsilon > 0$ . We use this meta-theorem to show the existence of an FPT time algorithm for the SUBSET TOKEN SWAPPING problem, restricted to nowhere-dense graph classes. In particular, this implies the following results.

► **Corollary 3.** *SUBSET TOKEN SWAPPING is FPT*

- (a) *parameterized by  $k + \text{tw}(G)$ ,*
- (b) *parameterized by  $k$  in planar graphs.*

It is often observed that NP-hard graph problems become tractable on classes of graphs with bounded treewidth (or, at least, with bounded tree-depth; see Nešetřil and Ossona de Mendez [26, Chapter 10] for the definition and some background of tree-depth and related parameters). It is not uncommon to see FPT algorithms running in time  $f(\text{tw})n^{O(1)}$  (or  $f(\text{td})n^{O(1)}$ ) or XP algorithms running in time  $n^{f(\text{tw})}$  (or  $n^{f(\text{td})}$ ), for some computable functions  $f$ . Especially, in light of Corollary 3(a), we want to know if there exists an algorithm that runs in polynomial time for constant treewidth. In Section 6 we rule out the existence of such algorithms by showing that TOKEN SWAPPING remains NP-hard when restricted to graphs with tree-depth 4 (treewidth and pathwidth 2; diameter 6; distance 1 to a forest).

► **Theorem 4** (Hard on Almost Trees). *TOKEN SWAPPING remains NP-hard even when both the treewidth and the diameter of the input graph are constant, and cannot be solved in time  $2^{o(n)}$ , unless the ETH fails.*

The Table 1 shows the current state of our knowledge about the parameterized complexity of TOKEN SWAPPING (TS), COLORED TOKEN SWAPPING (CTS), and SUBSET TOKEN SWAPPING (STS) problems, for different choices of parameters.



■ **Table 2** The complexity of TOKEN SWAPPING (TS), COLORED TOKEN SWAPPING (CTS), and SUBSET TOKEN SWAPPING (STS) on very restricted classes of graphs. The results in bold are proved in this paper. The three polynomial algorithms for TOKEN SWAPPING on cliques, stars, and paths, are folklore and can be found for instance in [25, 19].

	trees	cliques	stars	paths
TS	?	P	P	P
CTS	?	<b>NP-c</b>	<b>P</b>	<b>P</b>
STS	<b>NP-c</b>	<b>NP-c</b>	<b>NP-c</b>	?

While we think that our results give a fairly detailed view on the complexity landscape of the TOKEN SWAPPING problem, we also want to point out that our reductions are significantly simpler than those by Miltzow *et al.* [25].

Since the investigated problems seem to be immensely intractable, we investigate their complexities in very restricted classes of graphs, namely cliques, stars, and paths. We focus on finding the borderlines between easy (polynomially solvable) and hard (NP-hard) cases. The summary of these results is given in Table 2. Observe that cliques distinguish the complexities of the TOKEN SWAPPING and the COLORED TOKEN SWAPPING problems, while stars distinguish the complexities of the COLORED TOKEN SWAPPING and the SUBSET TOKEN SWAPPING problems.

The paper concludes with several open problems in Section 7.

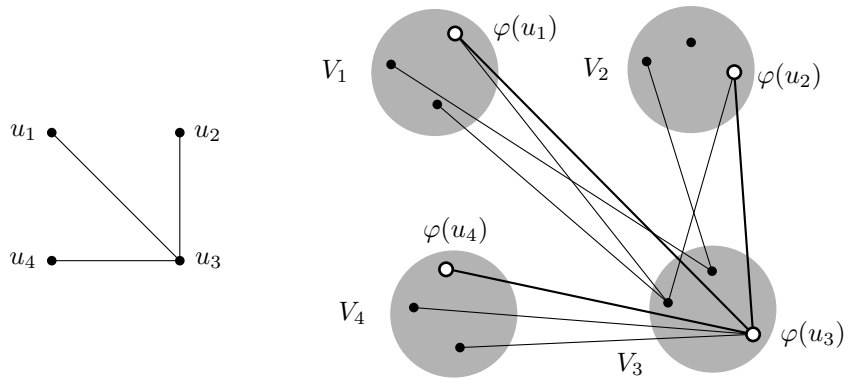
## 2 Preliminaries

For a token  $t$ , let  $\text{dist}(t)$  denote the distance from the position of  $t$  to its destination. For an instance  $I$  of the TOKEN SWAPPING problem, we define  $L(I) := \sum_t \text{dist}(t)$ , i.e., the sum of distances to the destination over all the tokens. Clearly, after performing a single swap,  $\text{dist}(t)$  may change by at most 1. We shall also use the following classification of swaps: for  $x, y \in \{-1, 0, 1\}$ ,  $x \leq y$ , by a  $(x, y)$ -swap we mean a swap, in which one token changes its distance by  $x$ , and the other one by  $y$ . Intuitively,  $(-1, -1)$ -swaps are the most “efficient” ones, thus we will call them *happy swaps*. Since each swap involves two tokens, we get the following lower bound.

▶ **Proposition 5** ([25]). *The length of an optimal solution for an instance  $I$  of TOKEN SWAPPING is at least  $L(I)/2$ . Besides, it is exactly  $L(I)/2$  iff there is a solution using happy swaps only.*

When designing algorithms, it is natural to ask about lower bounds. However, the standard complexity assumption used for distinguishing easy and hard problems,  $P \neq NP$ , cannot rule out, say, subexponential time algorithms. The stronger assumption that is typically used for this purpose is the so-called *Exponential Time Hypothesis* (ETH), formulated by Impagliazzo and Paturi [17]. We refer the reader to the survey by Lokshtanov and Marx for more information about ETH and conditional lower bounds [22]. The version we present below (and is most commonly used) is not the original statement of this hypothesis, but its weaker version (see also Impagliazzo, Paturi, and Zane [18]).

▶ **Exponential Time Hypothesis** (Impagliazzo and Paturi [17]). *There is no algorithm solving every instance of 3-SAT with  $N$  variables and  $M$  clauses in time  $2^{o(N+M)}$ .*



■ **Figure 2** On the left is the pattern graph  $P$ ; on the right, the host graph  $H$ . We indicate the image of  $\varphi$  with white vertices. To keep the example small, we did not make  $P$  3-regular.

### 3 Algorithms

First, we prove that SUBSET TOKEN SWAPPING (and therefore also COLORED TOKEN SWAPPING as its restriction) is FPT in  $k + \Delta$ , where  $k$  is the number of allowed swaps, and  $\Delta$  is the maximum degree of the input graph. This generalizes the observation of Miltzow *et al.* [25] for the TOKEN SWAPPING problem. Furthermore, we show that the simple algorithm for the TOKEN SWAPPING problem, presented by Miltzow *et al.* [25], carries over to the generalized problems, i.e., COLORED TOKEN SWAPPING and SUBSET TOKEN SWAPPING. At last, we will present an algorithm that has polynomial space complexity.

► **Proposition 6.** SUBSET TOKEN SWAPPING problem is FPT in  $k + \Delta$  and admits a kernel of size  $2k + 2k^2 \cdot \Delta^k$ .

Miltzow *et al.* [25] show that an optimal solution for the TOKEN SWAPPING problem can be found by performing a breath-first-search on the *configuration graph*, that is, the graph whose vertices are all possible configurations of tokens on vertices, and two configurations are adjacent when one can be obtained from the other with a single swap<sup>1</sup>. We observe that the same approach works for the COLORED TOKEN SWAPPING and the SUBSET TOKEN SWAPPING problems, the only difference is that we terminate on any feasible target configuration.

The main drawback of such an approach is an exponential space complexity. Here we show the following complementary result, inspired by the ideas of Savitch [29].

► **Theorem 7.** Let  $G$  be a graph with  $n$  vertices, and let  $k$  be the maximum number of allowed swaps. The SUBSET TOKEN SWAPPING problem on  $G$  can be solved in time  $2^{O(n \log n \log k)} = 2^{O(n \log^2 n)}$  and space  $O(n \log n \log k) = O(n \log^2 n)$ .

### 4 Lower Bounds on parameterized Token Swapping

Let us start by defining an auxiliary problem, called MULTICOLORED SUBGRAPH ISOMORPHISM (also known as PARTITIONED SUBGRAPH ISOMORPHISM; see Figure 2).

<sup>1</sup> The configuration graph of a TOKEN SWAPPING instance on a graph  $G$  with  $n$  vertices can also be seen as the Cayley graph  $\Gamma(\mathcal{P}_n, S)$  where  $\mathcal{P}_n$  is the symmetric group on  $n$  elements and  $S$  is the set of all transpositions  $(u v)$  where  $uv$  is an edge of  $G$ .

In MULTICOLORED SUBGRAPH ISOMORPHISM, one is given a host graph  $H$  whose vertex set is partitioned into  $k$  color classes  $V_1 \uplus V_2 \uplus \dots \uplus V_k$  and a pattern graph  $P$  with  $k$  vertices:  $V(P) = \{u_1, \dots, u_k\}$ . The goal is to find an injection  $\varphi : V(P) \rightarrow V(H)$  such that  $u_i u_j \in E(P)$  implies that  $\varphi(u_i) \varphi(u_j) \in E(H)$  and  $\varphi(u_i) \in V_i$  for all  $i, j$ . Thus we can assume that each  $V_i$  forms an independent set. Further, we assume without loss of generality that  $E(V_i, V_j) := \{ab \in E(H) : a \in V_i, b \in V_j\}$  is non-empty iff  $u_i u_j \in E(P)$ . In other words, we try to find  $k$  vertices  $v_1 \in V_1, v_2 \in V_2, \dots, v_k \in V_k$  such that, for any  $i < j \in [k]$ ,<sup>2</sup> there is an edge between  $v_i$  and  $v_j$  iff  $E(V_i, V_j)$  is non-empty. The  $W[1]$ -hardness of MULTICOLORED SUBGRAPH ISOMORPHISM problem follows from the  $W[1]$ -hardness of the MULTICOLORED CLIQUE. Marx [23] showed that assuming the ETH, MULTICOLORED SUBGRAPH ISOMORPHISM cannot be solved in time  $f(k)(|V(H)| + |E(H)|)^{o(k/\log k)}$ , for any computable function  $f$ , even when the pattern graph  $P$  is 3-regular and bipartite (see also Marx and Pilipczuk [24]). In particular,  $k$  has to be an even integer since  $|E(P)|$  is exactly  $3k/2$ . We finally assume that for every  $i \in [k]$  it holds that  $|V_i| = t$ , by padding potentially smaller classes with isolated vertices. This can only increase the size of the host graph by a factor of  $k$ , and does not create any new solution nor destroy any existing one.

Now we are ready to prove the following theorem.

► **Theorem 1 (Parameterized Hardness).** *TOKEN SWAPPING is  $W[1]$ -hard, parameterized by the number  $k$  of allowed swaps. Moreover, assuming the ETH, for any computable function  $f$ , TOKEN SWAPPING cannot be solved in time  $f(k)(n + m)^{o(k/\log k)}$  where  $n$  and  $m$  are respectively the number of vertices and edges of the input graph.*

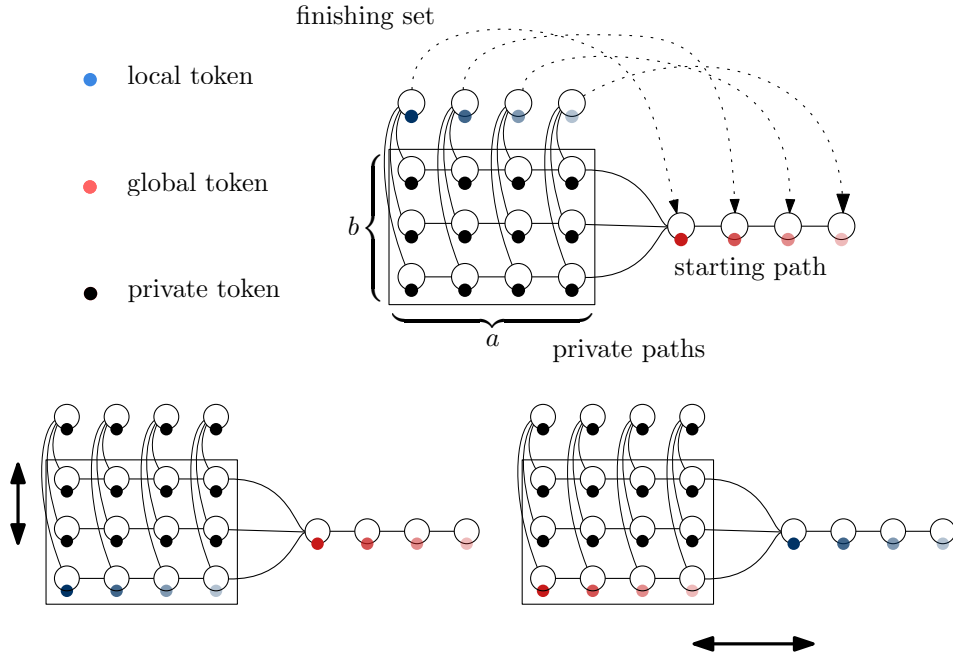
**Proof (Sketch).** We will present a reduction from MULTICOLORED SUBGRAPH ISOMORPHISM. To show the parameterized hardness of the TOKEN SWAPPING problem, we introduce a very handy *linker gadget*. This gadget has a robust and general ability to link decisions. As such, it permits to reduce from a wide range of problems. Its description is short and its soundness is intuitive. Because it yields very light constructions, we can rule out fairly easily unwanted swap sequences. We describe the linker gadget and provide some intuitive reason why it works (see Figure 3).

**Linker gadget.** Given two integers  $a$  and  $b$ , the linker gadget  $L_{a,b}$  contains a set of  $a$  vertices, called *finishing set* and a path on  $a$  vertices, that we call *starting path*. The tokens initially on vertices of the finishing set are called *local tokens*; they shall go to the vertices of the starting path in the way depicted in Figure 3. The tokens initially on vertices of the starting path are called *global tokens*. Global tokens have their destination in some other linker gadget. To be more specific, their destination is in the finishing set of another linker.

We describe and always imagine the finishing set and the starting paths *to be ordered from left to right*. Below the finishing set and to the left of the starting path, stand  $b$  disjoint induced paths, each with  $a$  vertices, arranged in a grid, see Figure 3. We call those paths *private paths*. The *private tokens* on private paths are already well-placed. Every vertex in the finishing set is adjacent to all private vertices below it and the leftmost vertex of the starting path is adjacent to all rightmost vertices of the private paths.

For local tokens to go to the starting path, they must go through a private path. As its name suggests, the linker gadget aims at linking the choice of the private path used for every local token. Intuitively, the only way of benefiting from  $a^2$  happy swaps between the  $a$  local tokens and the  $a$  global tokens is to use a common private path (note that the destination

<sup>2</sup> For an integer  $p$ , by  $[p]$  we denote the set  $\{1, \dots, p\}$ .



■ **Figure 3** The linker gadget  $L_{a,b}$ . Black tokens are initially properly placed. Dashed arcs represent where tokens of the finishing set should go in the starting path. At the bottom left, we depict the gadget after all the local tokens are swapped to a single private path. At the bottom right, we see the result after swapping all the local tokens to the starting path. In this case, the global tokens go to that private path.

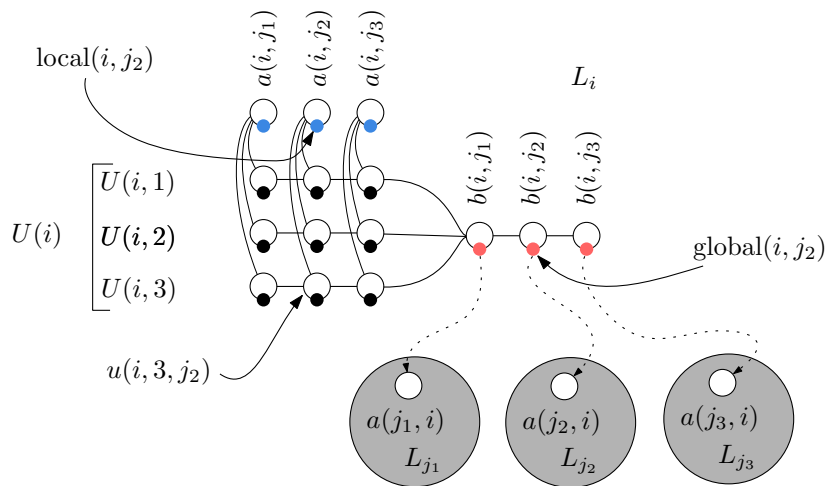
of the global tokens will make those swaps happy). That results in a kind of configuration as depicted in the bottom right of Figure 3, where each global token is in the same private path. The fate of the global tokens has been linked.

**Construction.** We present a reduction from MULTICOLORED SUBGRAPH ISOMORPHISM with cubic pattern graphs to TOKEN SWAPPING where the number of allowed swaps is linear in  $k$ . Let  $(H, P)$  be an instance of MULTICOLORED SUBGRAPH ISOMORPHISM. For any color class  $V_i = \{v_{i,1}, v_{i,2}, \dots, v_{i,t}\}$  of  $H$ , we add a copy of the linker  $L_{3,t}$  that we denote by  $L_i$ . We denote by  $j_1 < j_2 < j_3$  the indices of the neighbors of  $u_i$  in the pattern graph  $P$ . The linker  $L_i$  will be linked to 3 other gadgets and it has  $t$  private paths (or *choices*). The finishing set of  $L_i$  contains, from left to right, the vertices  $a(i, j_1)$ ,  $a(i, j_2)$ , and  $a(i, j_3)$ . We denote the tokens initially on the vertices  $a(i, j_1)$ ,  $a(i, j_2)$ , and  $a(i, j_3)$  by  $\text{local}(i, j_1)$ ,  $\text{local}(i, j_2)$ ,  $\text{local}(i, j_3)$ , respectively.

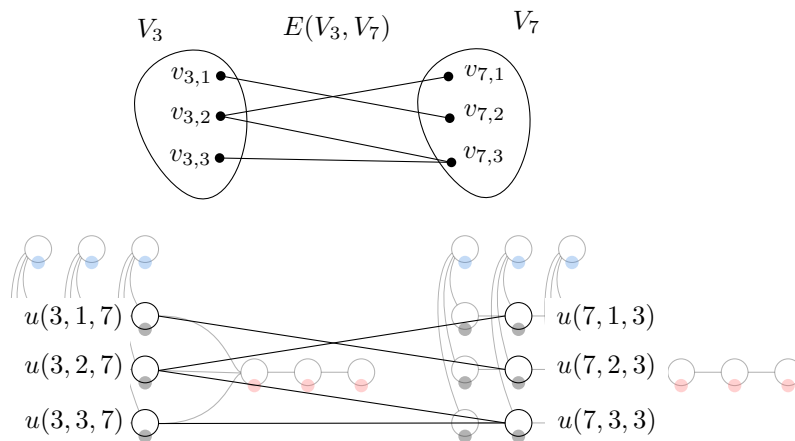
The starting path contains, from left to right, vertices  $b(i, j_1)$ ,  $b(i, j_2)$ , and  $b(i, j_3)$  with tokens  $\text{global}(i, j_1)$ ,  $\text{global}(i, j_2)$ , and  $\text{global}(i, j_3)$ .

For each  $p \in [3]$ ,  $\text{local}(i, j_p)$  shall go to vertex  $b(i, j_p)$ , whereas  $\text{global}(i, j_p)$  shall go to  $a(j_p, i)$  in the gadget  $L_{j_p}$ . Observe that the former transfer is internal and may remain within the gadget  $L_i$ , while the latter requires some interplay between the gadgets  $L_i$  and  $L_{j_p}$ . For any  $h \in [t]$ , by  $U(i, h)$  we denote the  $h$ -th private path. This path represents the vertex  $v_{i,h}$ . The path  $U(i, h)$  consists of, from left to right, vertices  $u(i, h, j_1)$ ,  $u(i, h, j_2)$ ,  $u(i, h, j_3)$ . We set  $U(i) := \bigcup_{h \in [t]} U(i, h)$ . Initially, all the tokens placed on vertices of  $U(i)$  are already well placed.

We complete the construction by adding every edge of the form  $u(i, h, j)u(j, h', i)$  if  $v_{i,h}v_{j,h'}$  is an edge in  $E(V_i, V_j)$  (see Figure 5). Let  $G$  be the graph that we built, and let  $I$



■ **Figure 4** The different labels for tokens, vertices, and sets of vertices.



■ **Figure 5** The way linkers (in that case,  $L_3$  and  $L_7$ ) are assembled together, with  $t = 3$ .

be the whole instance of TOKEN SWAPPING (with the initial position of the tokens). We claim that  $(H, P)$  is a YES-instance of MULTICOLORED SUBGRAPH ISOMORPHISM if and only if  $I$  has a solution of length at most  $\ell := 16.5k = O(k)$ . Recall that  $k$  is even, so  $16.5k$  is an integer.

**Correctness.** As already described above, the local tokens can reach their target vertices more efficiently if they all use the same private path. This private path represents a choice of the corresponding vertex in the original MULTICOLORED SUBGRAPH ISOMORPHISM instance. Thereafter, each global token can go with a single swap to its correct target gadget, if there were an edge between the corresponding vertices. This sequence needs exactly  $16.5k$  swaps.

The reverse direction is more difficult. Observe that, except from the swaps involving private tokens, all the swaps in the described solution are happy. However, it can be shown that the swaps that are not happy are necessary. In particular, any deviation from the intended solution requires additional swaps. ◀

## 5 Token Swapping on nowhere-dense classes of graphs

As we have seen in Section 4, there is little hope for an FPT algorithm for the TOKEN SWAPPING problem (parameterized by  $k$ ), unless  $\text{FPT} = W[1]$ . Now let us show that FPT algorithms exist, if we restrict our input to nowhere-dense graph classes.

The formal definition of nowhere-dense graphs is technical, so we refer the reader to the comprehensive book of Nešetřil and Ossona de Mendez [26, Chapter 13].

As graphs with bounded degree are nowhere-dense, this result generalizes Proposition 6.

► **Theorem 2** (FPT in nowhere dense graphs). *SUBSET TOKEN SWAPPING is FPT parameterized by  $k$  on nowhere-dense graph classes.*

We derive the following corollary.

► **Corollary 8.** *SUBSET TOKEN SWAPPING is FPT*

- (a) *parameterized by  $k + \text{tw}(G)$ ,*
- (b) *parameterized by  $k$  in planar graphs.*

To see Corollary 3 (a), recall that bounded-treewidth graphs are nowhere-dense. Thus by Theorem 2 there exists an algorithm with running time  $O(f(k) n^{1+\varepsilon})$ , for any  $\varepsilon > 0$  and treewidth bounded by some constant  $c$ . Observe that the constant hidden in the big-O notation depends on the constant  $c$ . In particular  $c$  has no influence on the exponent of  $n$ .

## 6 Token Swapping on almost trees

This section is devoted to the proof of the following theorem.

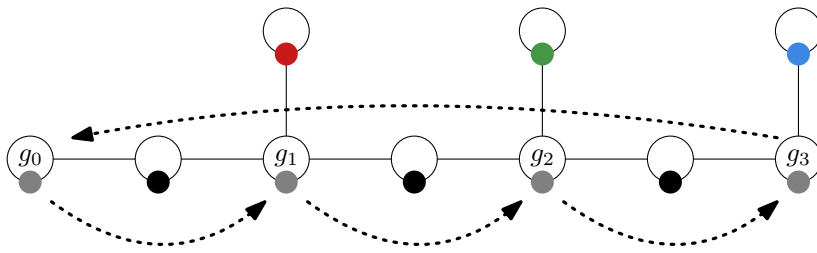
► **Theorem 4** (Hard on Almost Trees). *TOKEN SWAPPING remains NP-hard even when both the treewidth and the diameter of the input graph are constant, and cannot be solved in time  $2^{o(n)}$ , unless the ETH fails.*

**Proof.** In EXACT COVER BY 3-SETS, one is given a family  $\mathcal{S} = \{S_1, S_2, \dots, S_m\}$  of 3-element subsets of the universe  $X = \{x_1, x_2, \dots, x_n\}$ , where 3 divides  $n$ . The goal is to find  $n/3$  subsets in  $\mathcal{S}$  that partition (or here, equivalently, cover)  $X$ . The problem can be seen as a straightforward generalization of the 3-DIMENSIONAL MATCHING problem. This problem is NP-complete and has no  $2^{o(n)}$  algorithm, unless the ETH fails, even if each element belongs to exactly 3 triples [13, 3]. Therefore we can reduce from the restriction of the EXACT COVER BY 3-SETS problem, where each element belongs to 3 sets of  $\mathcal{S}$ , and obviously  $|\mathcal{S}| = |X| = n$ .

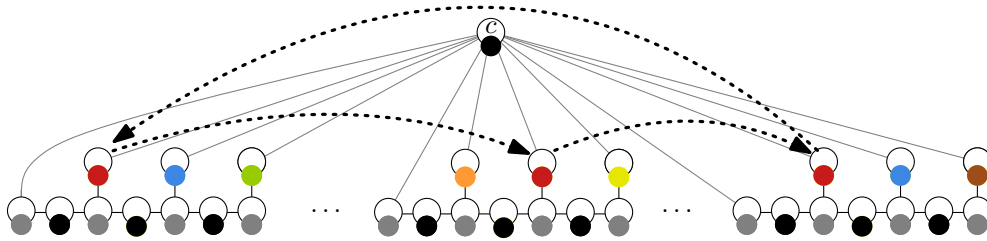
**Construction.** For each set  $S_j \in \mathcal{S}$ , we add a *set gadget* consisting of a tree on 10 vertices (see Figure 6). In the set gadget, the four gray tokens should cyclically swap as indicated by the dotted arrows:  $g_i^j$  shall go where  $g_{i+1}^j$  is, for each  $i \in [4]$  (addition is computed modulo 4). The three black tokens, as usual, are initially well placed. The three remaining vertices are called *element* vertices. They represent the three elements of the set. The tokens initially on the *element* vertices are called *element* tokens. For each element of  $X$ , there are 3 *element* tokens and 3 *element* vertices.

We add a vertex  $c$  that is linked to all the *element* vertices of the set gadgets and to all the vertices  $g_0^j$ . Each token originally on an *element* vertex should cyclically go to *its next occurrence* (see Figure 7). The token initially on  $c$  is well placed.

The constructed graph  $G$  has  $10n + 1$  vertices. If one removes the vertex  $c$  the remaining graph is a forest, which means that the graph has a feedback vertex set of size 1 and, in



■ **Figure 6** The set gadget for red, green and blue. We voluntarily omit the superscript  $j$ .



■ **Figure 7** The overall picture. Each element appears exactly 3 times, so there are 3 red tokens.

particular, treewidth 2.  $G$  has its diameter bounded by 6, since all the vertices are at distance at most 3 of the vertex  $c$ . We now show that the instance  $\mathcal{S}$  of EXACT COVER BY 3-SETS admits a solution iff there exists a solution for our instance of TOKEN SWAPPING of length at most  $\ell := 11 \cdot n/3 + 9 \cdot 2n/3 + 2n = 35n/3 = 11n + 2n/3$ .

**Soundness.** The correctness of the construction relies mainly on the fact that there are two competitive ways of placing the gray tokens. The first way is the most direct. It consists of only swapping along the *spine* of the set gadget. By *spine*, we mean the 7 vertices initially containing gray or black tokens. From hereon, we call that *swapping the gray tokens internally*.

► **Claim 9.** *Swapping the gray tokens internally requires 9 swaps.*

**Proof.** In 6 swaps, we can first move  $g_3$  to its destination (where  $g_0$  is initially). Then,  $g_0$ ,  $g_1$ , and  $g_2$  need one additional swap each to be correctly placed. We observe that, after we do so, the black tokens are back to their respective destination. ◀

We call the second way *swapping the gray tokens via  $c$* . Basically, it is the way one would have to place the gray tokens if the black tokens (except the one in  $c$ ) were removed from the graph. It consists of, first (a) swapping  $g_0$  with the token on  $c$ , then moving  $g_0$  to its destination, then (b) swapping  $g_1$  with the current token on  $c$ , moving  $g_1$  to its destination, (c) swapping  $g_2$  with the token on  $c$ , moving  $g_2$  to its destination, finally (d) swapping  $g_3$  with the token on  $c$  and moving it to its destination.

► **Claim 10.** *Swapping the gray tokens via  $c$  requires 11 swaps.*

**Proof.** Steps (a), (b), and (c) take 3 swaps each, while step (d) takes 2 swaps. ◀

Considering that swapping the gray tokens via  $c$  takes 2 more swaps than swapping them internally, and leads to the exact same configuration where both the black tokens and the *element* tokens are back to their initial position, one can question the interest of the second

way of swapping the gray tokens. It turns out that, at the end of steps (a), (b), and (c), an *element* token is on vertex  $c$ . We will take advantage of that situation to perform two consecutive happy swaps with its two other occurrences. By doing so, observe that the first swap of steps (b), (c), and (d) are also happy and place the last occurrence of the *element* tokens at its destination.

We assume that there is a solution  $S_{a_1}, \dots, S_{a_{n/3}}$  to the EXACT COVER BY 3-SETS instance. In the corresponding  $n/3$  set gadgets, swap the gray tokens via  $c$  and interleave those swaps with doing the two happy swaps over *element* tokens, whenever such a token reaches  $c$ . By Claim 10, this requires  $11 \cdot n/3 + 2n$  swaps. At this point, the tokens that are misplaced are the  $4 \cdot 2n/3$  gray tokens in the  $2n/3$  remaining set gadgets. Swap those gray tokens internally. This adds  $9 \cdot 2n/3$  swaps, by Claim 9. Overall, this solution consists of  $29n/3 + 2n = 35n/3 = \ell$ .

Let us now suppose that there is a solution  $\mathbf{s}$  of length at most  $\ell$  to the TOKEN SWAPPING instance. At this point, we should observe that there are alternative ways (to Claim 9 and Claim 10) of placing the gray tokens at their destination. For instance, one can move  $g_3$  to  $g_1$  along the spine, place tokens  $g_2$  and  $g_3$ , then exchange  $g_0$  with the token on  $c$ , move  $g_0$  to its destination, swap  $g_3$  with the token on  $c$ , and finally move it to its destination. This also takes 11 swaps but moves only one *element* token to  $c$  (compared to moving all three of them in the strategy of Claim 10). One can check that all those alternative ways take 11 swaps or more. Let  $r \in [0, n]$  be such that  $\mathbf{s}$  does *not* swap the gray tokens internally in  $r$  set gadgets (and swap them internally in the remaining  $n - r$  set gadgets). The length of  $\mathbf{s}$  is at least  $11r + 9(n - r) + 2(n - q) + 4q = 11n + 2(r + q)$ , where  $q$  is the number of elements of  $X$  for which *none* occurrence of its three *element* tokens has been moved to  $c$  in the process of swapping the gray tokens. Indeed, for each of those  $q$  elements, 4 additional swaps will be eventually needed. For each of the remaining  $n - q$  elements, only 2 additional happy swaps will place the three corresponding *element* tokens at their destination. It holds that  $3r \geq n - q$ , since the *element* tokens within the  $r$  set gadgets where  $\mathbf{s}$  does not swap internally represent at most  $3r$  distinct elements of  $X$ . Hence,  $3r + q \geq n$ . Also,  $\mathbf{s}$  is of length at most  $\ell = 11n + 2n/3$ , which implies that  $r + q \leq n/3$ . Thus,  $n \leq 3r + q \leq 3r + 3q \leq n$ . Therefore,  $q = 0$  and  $r = n/3$ . Let  $S_{a_1}, \dots, S_{a_{n/3}}$  be the  $n/3$  sets for which  $\mathbf{s}$  does not swap the gray tokens internally in the corresponding set gadgets. For each element of  $X$ , an occurrence of a corresponding *element* token is moved to  $c$  when the gray tokens are swapped in one of those gadgets. So this element belongs to one  $S_{a_i}$  and therefore  $S_{a_1}, \dots, S_{a_{n/3}}$  is a solution to the instance of EXACT COVER BY 3-SETS.

The ETH lower bound follows from the fact, that the size of constructed graph is  $O(n)$ . ◀

## 7 Conclusion

We conclude the paper with several ideas for further research. First, we believe that it would be interesting to fill the missing entries in Table 2. In particular, we conjecture that the TOKEN SWAPPING problem remains NP-complete even if the input graph is a tree.

Another interesting problem is the following. By Miltzow *et al.* [25, Theorem 1], the TOKEN SWAPPING problem can be solved in time  $2^{O(n \log n)}$ , and there is no  $2^{o(n)}$  algorithm, unless the ETH fails. We conjecture that the lower bound can be improved to  $2^{o(n \log n)}$ . It would also be interesting to find single-exponential algorithms for some restricted graph classes, such as graphs with bounded treewidth or planar graphs.

Finally, to prove Corollary 3, we use the powerful and very general meta-theorem by Grohe, Kreutzer, and Siebertz [15]. It would be interesting to obtain elementary FPT algorithms for planar graphs and graphs with bounded treewidth (or even trees).



---

**References**

---

- 1 Sheldon B Akers and Balakrishnan Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *IEEE transactions on Computers*, 38(4):555–566, 1989.
- 2 Elwyn R Berlekamp, John Horton Conway, and Richard K Guy. *Winning Ways, for Your Mathematical Plays: Games in particular*, volume 2. Academic Pr, 1982.
- 3 Hans L. Bodlaender and Jesper Nederlof. Subexponential time algorithms for finding small tree and path decompositions. In *ESA 2015 Proc.*, pages 179–190. Springer, 2015. doi:10.1007/978-3-662-48350-3\_16.
- 4 Prosenjit Bose and Ferran Hurtado. Flips in planar graphs. *Computational Geometry*, 42(1):60–80, 2009.
- 5 Gruia Calinescu, Adrian Dumitrescu, and János Pach. Reconfigurations in graphs and grids. In *LATIN 2006 Proc.*, pages 262–273. Springer, 2006. doi:10.1007/11682462\_27.
- 6 Arthur Cayley. LXXVII. Note on the theory of permutations. *Philosophical Magazine Series 3*, 34(232):527–529, 1849. doi:10.1080/14786444908646287.
- 7 Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 2000.
- 8 Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Linear-time algorithm for sliding tokens on trees. *Theoretical Computer Science*, 600:132–142, 2015. doi:10.1016/j.tcs.2015.07.037.
- 9 Ruy Fabila-Monroy, David Flores-Peñaloza, Clemens Huemer, Ferran Hurtado, Jorge Urrutia, and David R. Wood. Token graphs. *Graphs and Combinatorics*, 28(3):365–380, 2012. doi:10.1007/s00373-011-1055-9.
- 10 F. Farnoud, C. Y. Chen, O. Milenkovic, and N. Kashyap. A graphical model for computing the minimum cost transposition distance. In *Information Theory Workshop (ITW), 2010 IEEE*, pages 1–5, Aug 2010. doi:10.1109/CIG.2010.5592890.
- 11 Farzad Farnoud and Olgica Milenkovic. Sorting of permutations by cost-constrained transpositions. *IEEE Trans. Information Theory*, 58(1):3–23, 2012. doi:10.1109/TIT.2011.2171532.
- 12 Eli Fox-Epstein, Duc A. Hoang, Yota Otachi, and Ryuhei Uehara. Sliding token on bipartite permutation graphs. In Khaled Elbassioni and Kazuhisa Makino, editors, *Algorithms and Computation*, volume 9472 of *Lecture Notes in Computer Science*, pages 237–247. Springer Berlin Heidelberg, 2015. doi:10.1007/978-3-662-48971-0\_21.
- 13 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.
- 14 Daniel Graf. How to sort by walking on a tree. In *ESA 2015 Proc.*, pages 643–655. Springer, 2015. doi:10.1007/978-3-662-48350-3\_54.
- 15 Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. In *STOC 2014 Proc.*, pages 89–98. ACM, 2014. doi:10.1145/2591796.2591851.
- 16 Lenwood S. Heath and John Paul C. Vergara. Sorting by short swaps. *Journal of Computational Biology*, 10(5):775–789, 2003. doi:10.1089/106652703322539097.
- 17 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.
- 18 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.
- 19 Mark R. Jerrum. The complexity of finding minimum-length generator sequences. *Theoretical Computer Science*, 36:265–289, 1985. doi:10.1016/0304-3975(85)90047-7.

- 20 Takumi Kasai, Akeo Adachi, and Shigeki Iwata. Classes of pebble games and complete problems. *SIAM Journal on Computing*, 8(4):574–586, 1979. doi:10.1137/0208046.
- 21 Donald Erwin Knuth. *The Art of Computer Programming*, volume 3 / Sorting and Searching. Addison-Wesley, 1982. ISBN 0-201-03803-X.
- 22 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011. URL: <http://albcm.lsi.upc.edu/ojs/index.php/beatcs/article/view/96>.
- 23 Dániel Marx. Can you beat treewidth? *Theory of Computing*, 6(1):85–112, 2010. doi:10.4086/toc.2010.v006a005.
- 24 Dániel Marx and Michal Pilipczuk. Optimal parameterized algorithms for planar facility location problems using voronoi diagrams. *CoRR*, abs/1504.05476, 2015. URL: <http://arxiv.org/abs/1504.05476>.
- 25 Tillmann Miltzow, Lothar Narins, Yoshio Okamoto, Günter Rote, Antonis Thomas, and Takeaki Uno. Approximation and hardness of token swapping. 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 66:1–66:15. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPICs.ESA.2016.66.
- 26 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity – Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-27875-4.
- 27 Igor Pak. Reduced decompositions of permutations in terms of star transpositions, generalized Catalan numbers and k-ARY trees. *Disc. Math.*, 204(1):329–335, 1999. doi:10.1016/S0012-365X(98)00377-X.
- 28 Torrence D Parsons. Pursuit-evasion in a graph. In *Theory and applications of graphs*, pages 426–441. Springer, 1978.
- 29 Walter J. Savitch. Relationships Between Nondeterministic and Deterministic Tape Complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.
- 30 Richard M. Wilson. Graph puzzles, homotopy, and the alternating group. *Journal of Combinatorial Theory, Series B*, 16(1):86–96, 1974. doi:10.1016/0095-8956(74)90098-7.
- 31 Katsuhisa Yamanaka, Erik D. Demaine, Takehiro Ito, Jun Kawahara, Masashi Kiyomi, Yoshio Okamoto, Toshiki Saitoh, Akira Suzuki, Kei Uchizawa, and Takeaki Uno. Swapping labeled tokens on graphs. In *FUN 2014 Proc.*, pages 364–375. Springer, 2014. doi:10.1007/978-3-319-07890-8\_31.
- 32 Katsuhisa Yamanaka, Takashi Horiyama, David G. Kirkpatrick, Yota Otachi, Toshiki Saitoh, Ryuhei Uehara, and Yushi Uno. Swapping colored tokens on graphs. In *WADS 2015 Proc.*, pages 619–628, 2015. doi:10.1007/978-3-319-21840-3\_51.
- 33 Gaku Yasui, Kouta Abe, Katsuhisa Yamanaka, and Takashi Hirayama. Swapping labeled tokens on complete split graphs. *SIG Technical Reports*, 2015-AL-153(14):1–4, 2015.

# Monte Carlo Computability

Vasco Brattka<sup>\*1</sup>, Rupert Hölzl<sup>2</sup>, and Rutger Kuyper<sup>3</sup>

- 1 Department of Mathematics and Applied Mathematics, University of Cape Town, Cape Town, South Africa; and  
Faculty of Computer Science, Universität der Bundeswehr München, Neubiberg, Germany  
Vasco.Brattka@cca-net.de
- 2 Faculty of Computer Science, Universität der Bundeswehr München, Neubiberg, Germany  
r@hoelzl.fr
- 3 School of Mathematics and Statistics, Victoria University of Wellington, Wellington, New Zealand  
mail@rutgerkuyper.com

---

## Abstract

We introduce Monte Carlo computability as a probabilistic concept of computability on infinite objects and prove that Monte Carlo computable functions are closed under composition. We then mutually separate the following classes of functions from each other: the class of multi-valued functions that are non-deterministically computable, that of Las Vegas computable functions, and that of Monte Carlo computable functions. We give natural examples of computational problems witnessing these separations. As a specific problem which is Monte Carlo computable but neither Las Vegas computable nor non-deterministically computable, we study the problem of sorting infinite sequences that was recently introduced by Neumann and Pauly. Their results allow us to draw conclusions about the relation between algebraic models and Monte Carlo computability.

**1998 ACM Subject Classification** F.4.1 Mathematical Logic

**Keywords and phrases** Weihrauch degrees, Weak Weak König's Lemma, Monte Carlo computability, algorithmic randomness, sorting

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.17

## 1 Introduction

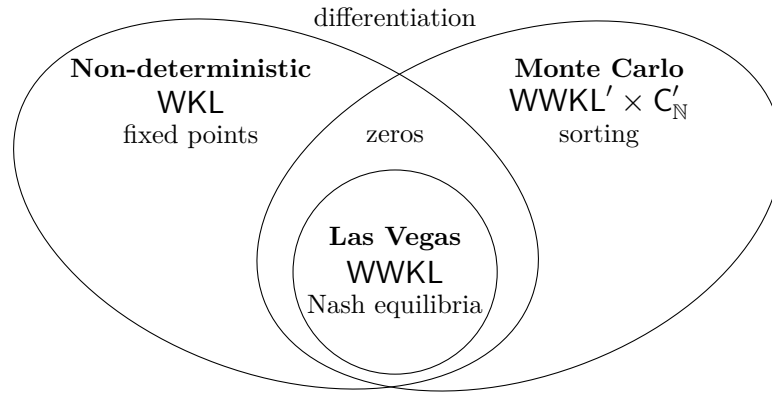
It is folklore in computational complexity theory that different machine models such as deterministic machines, non-deterministic machines and probabilistic machines describe the same classes of computable problems while they potentially yield different classes of polynomial-time computable problems. The complexity classes NP, BPP and ZPP are classes of decision problems which are polynomial-time computable on non-deterministic machines, Monte Carlo machines, and Las Vegas machines, respectively. The separation of these classes is a major and challenging problem in computational complexity theory.

In recent years it emerged that the situation for computations on infinite objects is somewhat different in that the classes of problems which are non-deterministically computable or probabilistically computable are actually strictly larger than the class of deterministically

---

\* Vasco Brattka has received funding from the National Research Foundation of South Africa. Rutger Kuyper has received funding from the John Templeton Foundation.





■ **Figure 1** Classes of non-deterministically and probabilistically computable problems.

computable problems from the mere point of view of computability theory even without time complexity considerations.

Computability over infinite objects is understood here in the well-established sense of computable analysis [28, 13] and the Weihrauch lattice [6, 24] is used as a fine-grained tool to express these results. Intuitively, it offers a notion of many-one reducibility for partial multi-valued functions  $f : \subseteq X \rightrightarrows Y$  that can be used to compare computational problems in a very natural and straightforward way.

Work on probabilistic notions in this setting has been started by Brattka and Pauly [15], Dorais, Dzhafarov, Hirst, Mileti and Shafer [18], Bienvenu and Porter [2] and others. In recent work Brattka, Gherardi and Hölzl [7, 8] have proposed the concept of Las Vegas computable functions and studied its computational power. The problem of computing Nash equilibria is an example of a Las Vegas computable problem, which is not deterministically computable.

Roughly speaking, a function  $f : \subseteq X \rightrightarrows Y$  is *Las Vegas computable* if it can be computed on a Turing machine upon input of (a name of) some  $x \in \text{dom}(f)$  with the help of an additional advice  $r \in 2^{\mathbb{N}}$  subject to the following conditions:

1. If the advice  $r \in 2^{\mathbb{N}}$  is not helpful, then the machine recognizes this in finite time and stops the computation with a failure signal.
2. If the advice  $r \in 2^{\mathbb{N}}$  is helpful, then the machine computes forever and produces a correct result, that is, (a name of) some  $y \in f(x)$ .
3. The set of helpful advices  $r$  for each fixed name of input  $x$  has to be of positive measure.

We continue this work in the present article and propose the new concept of Monte Carlo computability, which is different from Las Vegas computability in that we relax condition 1. Roughly speaking, we consider  $f : \subseteq X \rightrightarrows Y$  as *Monte Carlo computable* if the fact that the advice  $r \in 2^{\mathbb{N}}$  is not helpful can be recognized (only) in the limit. As a consequence of this relaxation such a machine might compute forever without producing a correct result. However, with positive probability it will produce a correct result.

In the Weihrauch lattice we can identify the classes WKL, WWKL and  $\text{WWKL}' \times C'_{\mathbb{N}}$  as being complete for non-deterministically computable, Las Vegas computable and Monte Carlo computable problems, respectively. All these classes are variants of Weak König's Lemma (WKL) and will be formally defined below. In a vague analogy these classes correspond to the complexity classes NP, ZPP and BPP, respectively.

In contrast to the situation in computational complexity theory, we can separate the classes  $WKL, WWKL$  and  $WWKL' \times C_{\mathbb{N}}'$  and provide natural computational problems as witnesses for these separations. The picture that emerges is illustrated in Figure 1. The problems given in the picture have been studied before:

1. Differentiation is the problem  $d: \subseteq \mathcal{C}[0, 1] \rightarrow \mathcal{C}[0, 1], f \mapsto f'$  to determine the derivative of a continuously differentiable function  $f: [0, 1] \rightarrow \mathbb{R}$  [27] and it is neither non-deterministically computable nor probabilistically computable in any form [8].
2. The fixed point problem  $BFT: \mathcal{C}([0, 1]^n, [0, 1]^n) \rightrightarrows [0, 1]^n, f \mapsto \{x: f(x) = x\}$ , that is, the problem of determining a fixed point of a continuous function  $f: [0, 1]^n \rightarrow [0, 1]^n$  for  $n \geq 2$  [14], is non-deterministically computable but not probabilistically computable in any form [8].  $BFT$  stands for “Brouwer Fixed Point Theorem”.
3. The zero problem  $IVT: \subseteq \mathcal{C}[0, 1] \rightrightarrows [0, 1], f \mapsto f^{-1}\{0\}$ , that is, the problem that maps every continuous function  $f: [0, 1] \rightarrow \mathbb{R}$  with  $f(0) \cdot f(1) < 0$  to one of its zeros [5], is non-deterministically computable, has a Monte Carlo algorithm, but is not Las Vegas computable [8].  $IVT$  stands for “Intermediate Value Theorem”.
4. Nash is the problem that maps a bi-matrix game  $(A, B) \in \mathbb{R}^{m \times n} \times \mathbb{R}^{m \times n}$  to one of its Nash equilibria [23] and is Las Vegas computable [8].
5. Sorting stands for the problem  $SORT_2: 2^{\mathbb{N}} \rightarrow 2^{\mathbb{N}}$  of sorting a binary sequence [22]. It is not non-deterministically computable but has a Monte Carlo algorithm, as we will show.

The given classifications of these problems (also illustrated in Figure 1) follow from results in the given references, except for the case of sorting, which we will precisely define and discuss in Section 5. This section contains the main technical contributions of this article. In Section 2 we start with recalling the definition of Weihrauch reducibility and some basic algebraic operations, followed by an introduction of the concept of Monte Carlo computability in Section 3. In Section 4 we discuss versions of Weak Weak König’s Lemma. We close this article with a brief discussion of algebraic computation models in Section 6.

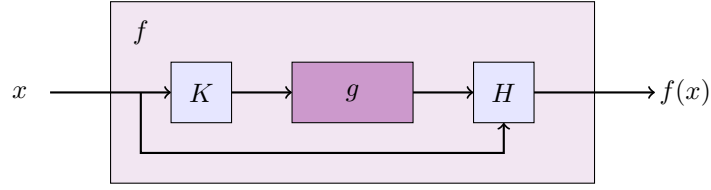
## 2 The Weihrauch Lattice

Formally, the Weihrauch lattice is formed by equivalence classes of partial multi-valued functions  $f: \subseteq X \rightrightarrows Y$  on represented spaces  $X, Y$ . We will simply call such functions *problems* here and they are, in fact, computational challenges in the sense that for every  $x \in \text{dom}(f)$  the goal is to find *some*  $y \in f(x)$ . In this case  $\text{dom}(f)$  contains the admissible instances  $x$  of the problem and for each instance  $x$  the set  $f(x)$  contains the corresponding solutions. Some typical problems  $f$  such as solving some type of equation or sorting a given sequence are mentioned above.

A *represented space*  $(X, \delta)$  is a set  $X$  together with a surjective partial map  $\delta: \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow X$  that assigns *names*  $p \in \mathbb{N}^{\mathbb{N}}$  to points  $\delta(p) = x \in X$ . These representations allow us to describe computations on all representable spaces using Turing machines that operate on the names corresponding to points in the space. We refer the reader to [28, 13] for details.

For problems  $f: \subseteq X \rightrightarrows Y$  and  $g: \subseteq Y \rightrightarrows Z$  we define the composition  $g \circ f: \subseteq X \rightrightarrows Z$  by  $g \circ f(x) = \{z \in Z: (\exists y \in f(x)) z \in g(y)\}$ , where  $\text{dom}(g \circ f) := \{x \in X: f(x) \subseteq \text{dom}(g)\}$ . We also denote the composition briefly by  $gf$ .

The intuition behind Weihrauch reducibility is that  $f \leq_W g$  holds if there is a computational procedure for solving  $f$  during which a single application of the computational resource  $g$  is allowed. There are actually two slightly different formal versions of this reduction, which are both needed. In expressions like  $H(x, gK(x))$  we tacitly use the definition of the composition as given above.



■ **Figure 2** Visualization of Weihrauch reducibility  $f \leq_W g$ .

- **Definition 1** (Weihrauch reducibility). Let  $f: \subseteq X \rightrightarrows Y$  and  $g: \subseteq W \rightrightarrows Z$  be problems.
1.  $f$  is called *Weihrauch reducible* to  $g$ , in symbols  $f \leq_W g$ , if there are computable  $K: \subseteq X \rightrightarrows W$ ,  $H: \subseteq X \times Z \rightrightarrows Y$  such that  $\emptyset \neq H(x, gK(x)) \subseteq f(x)$  for all  $x \in \text{dom}(f)$ .
  2.  $f$  is called *strongly Weihrauch reducible* to  $g$ , in symbols  $f \leq_{sW} g$ , if there are computable  $K: \subseteq X \rightrightarrows W$ ,  $H: \subseteq Z \rightrightarrows Y$  such that  $\emptyset \neq HgK(x) \subseteq f(x)$  for all  $x \in \text{dom}(f)$ .

The concept of Weihrauch reducibility is illustrated in Figure 2. The strong version of Weihrauch reducibility can be illustrated similarly without the direct input access of  $H$ .

Weihrauch reducibility induces a lattice with a rich and very natural algebraic structure. We briefly summarize some of these algebraic operations for problems  $f: \subseteq X \rightrightarrows Y$  and  $g: \subseteq W \rightrightarrows Z$ :

- $f \times g$  is the *product* of  $f$  and  $g$  and represents the parallel evaluation of problem  $f$  on some input  $x$  and  $g$  on some input  $w$ .
- $f * g := \sup\{f_0 \circ g_0 : f_0 \leq_W f \text{ and } g_0 \leq_W g\}$  is the *compositional product* and represents the consecutive usage of the problem  $f$  after the problem  $g$ .
- $f^* := \bigsqcup_{n=0}^{\infty} f^n$  is the *finite parallelization* and allows an evaluation of the  $n$ -fold product  $f^n$  for some arbitrary given  $n \in \mathbb{N}$ .
- $f'$  denotes the *jump* of  $f$ , which is formally the same problem, but the input representation  $\delta_X$  of  $X$  is replaced by its jump  $\delta'_X := \delta_X \circ \text{lim}$ .

Here  $\text{lim}: \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow \mathbb{N}^{\mathbb{N}}, \langle p_0, p_1, p_2, \dots \rangle \mapsto \lim_{i \rightarrow \infty} p_i$  is the usual limit map on Baire space for  $p_i \in \mathbb{N}^{\mathbb{N}}$ , where  $\langle \rangle$  denotes a standard infinite tupling function. One can also define a coproduct operation  $\sqcup$  and a sum operation  $\sqcap$  that play the role of supremum and infimum for ordinary Weihrauch reducibility  $\leq_W$ , respectively. But we are not going to use these operations here. The resulting Weihrauch lattice is not complete as infinite suprema do not need to exist, but the supremum  $f * g$  always exists as shown by Brattka and Pauly [16]. The finite parallelization is a closure operator in the Weihrauch lattice. Further information on the algebraic structure can be found in [16].

An important problem in the Weihrauch lattice is *closed choice*  $C_X: \subseteq \mathcal{A}_-(X) \rightrightarrows X$ ,  $A \mapsto A$ , which maps every closed set  $A \subseteq X$  to its points. The crucial fact here is that closed sets  $A \in \mathcal{A}_-(X)$  are represented with respect to negative information, essentially by enumerating open balls that exhaust their complement. That is, closed choice  $C_X$  is the following problem: given a closed set  $A$  by a description that lists everything that does not belong to  $A$ , find a point  $x \in A$  (see [4] for further information). We also consider  $PC_X$ , which is the restriction of  $C_X$  to sets of positive measure, where we assume that we have some given natural Borel measure on  $X$ . In case of Cantor space  $X = 2^{\mathbb{N}}$  we are going to use the uniform measure  $\mu_{2^{\mathbb{N}}}$ , in case of the reals  $X = \mathbb{R}$  we use the Lebesgue measure  $\mu_{\mathbb{R}}$ . Different classes of problems have been characterized by different forms of closed choice in the following ways [4, 8]:

- $f \leq_W C_{\mathbb{N}} \iff f$  is computable with finitely many mind changes.
- $f \leq_W C_{2^{\mathbb{N}}} \iff f$  is non-deterministically computable.
- $f \leq_W PC_{2^{\mathbb{N}}} \iff f$  is Las Vegas computable.

Here non-deterministic computability is understood in the way defined by Martin Ziegler [29] for the advice space  $2^{\mathbb{N}}$ .

### 3 Monte Carlo Computability

The notion of Las Vegas computability was introduced by Brattka, Gherardi and Hölzl [7, 8] and analogously we are going to introduce the notion of Monte Carlo computability here. The intuition of Monte Carlo computability was already described in the introduction and it is illustrated in Figure 3. The essential difference between Las Vegas and Monte Carlo computations is that in the former case we can recognize the failure of the advice in finite time, whereas in the latter case we can only recognize the failure in the limit. Indeed, one can even relax this condition further and obtain a whole hierarchy of notions of Monte Carlo computations, but we are not going to discuss this hierarchy here.

Instead of introducing Monte Carlo machines formally, we implicitly define them by using ordinary computable functions. Essentially, the computation of a Monte Carlo machine is governed by two functions  $F_1$  and  $F_2$ . Roughly speaking,  $F_1$  is responsible to provide the result of the computation upon some input and some additional advice  $r \in 2^{\mathbb{N}}$  and  $F_2$  is responsible to recognize whether the advice  $r$  fails.

The status of the advice is captured using Sierpiński space  $\mathbb{S} = \{0, 1\}$ , which is equipped with the topology  $\{\emptyset, \{1\}, \mathbb{S}\}$  and a corresponding representation  $\delta_{\mathbb{S}}$ . This space provides an asymmetric way to capture the status of the computation, very much in the same way as being computably enumerable is an asymmetric version of decidability. The point is that failure of an advice is supposed to be a recognizable event (in the limit), but success cannot necessarily be recognized in the same way. Monte Carlo computability can now be formalized as follows.

► **Definition 2** (Monte Carlo computability). Let  $(X, \delta_X)$  and  $(Y, \delta_Y)$  be represented spaces. A problem  $f: \subseteq X \rightrightarrows Y$  is said to be *Monte Carlo computable* if there exists a computable function  $F_1: \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow \mathbb{N}^{\mathbb{N}}$  and a limit computable function  $F_2: \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow \mathbb{S}$  such that  $\langle \text{dom}(f\delta_X) \times 2^{\mathbb{N}} \rangle \subseteq \text{dom}(F_2)$  and for each  $p \in \text{dom}(f\delta_X)$  the following hold:

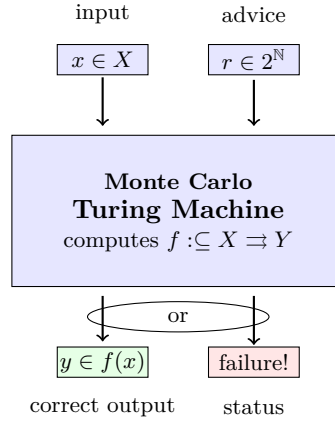
1.  $S_p := \{r \in 2^{\mathbb{N}} : F_2\langle p, r \rangle = 0\}$  is non-empty and  $\mu_{2^{\mathbb{N}}}(S_p) > 0$ ,
2.  $\delta_Y F_1\langle p, r \rangle \in f\delta_X(p)$  for all  $r \in S_p$ .

This difference to Las Vegas computability lies in the fact that  $F_2$  is only required to be limit computable (which is the same as effectively  $\Sigma_2^0$ -measurable) and not necessarily computable. While computable characteristic functions  $\chi_{2^{\mathbb{N}} \setminus A}: 2^{\mathbb{N}} \rightarrow \mathbb{S}$  capture exactly co-c.e. closed sets  $A$  (that is, effective  $\Pi_1^0$ -sets in the Borel hierarchy), limit computable characteristic functions  $\chi_{2^{\mathbb{N}} \setminus A}$  capture exactly effective  $G_\delta$ -sets  $A$  (that is, effective  $\Pi_2^0$ -sets in the Borel hierarchy) (see Pauly [25]). Hence, it should not come as a big surprise that Monte Carlo computability is closely related to choice for  $G_\delta$ -sets, which we define next. By  $\Pi_2^0(2^{\mathbb{N}})$  we denote the class of  $G_\delta$ -subsets of  $2^{\mathbb{N}}$ .

► **Definition 3** (Positive  $G_\delta$ -choice). By  $\Pi_2^0\text{PC}_{2^{\mathbb{N}}}: \subseteq \Pi_2^0(2^{\mathbb{N}}) \rightrightarrows 2^{\mathbb{N}}, A \mapsto A$  we denote the *positive  $G_\delta$ -choice problem*, defined for all  $A \in \Pi_2^0(2^{\mathbb{N}})$  with  $\mu_{2^{\mathbb{N}}}(A) > 0$ .

The crucial idea for a simple proof of the following result is to use a synthetic representation for the class of  $G_\delta$ -sets. There is a canonical function space representation  $[\delta_{2^{\mathbb{N}}} \rightarrow \delta_{\mathbb{S}}]$  of the continuous functions  $\chi: 2^{\mathbb{N}} \rightarrow \mathbb{S}$  and likewise  $[\delta_{2^{\mathbb{N}}} \rightarrow \delta'_{\mathbb{S}}]$  is a representation of the  $\Sigma_2^0$ -measurable functions [25] that we are going to use to represent  $\Pi_2^0(2^{\mathbb{N}})$ . This synthetic representation enables us to apply the methods of evaluation and type conversion. Hence we

## 17:6 Monte Carlo Computability



■ **Figure 3** Illustration of a Monte Carlo machine that computes  $f: \subseteq X \rightrightarrows Y$ .

can transfer the proof of [4, Theorem 7.2] by Brattka, de Brecht and Pauly literally to our setting.

► **Theorem 4** (Monte Carlo computability).  $f \leq_W \Pi_2^0 PC_{2^{\mathbb{N}}}$  if and only if  $f$  is Monte Carlo computable.

The class  $\Pi_2^0 PC_{2^{\mathbb{N}}}$  has been studied before and the following theorem was proved by Brattka, Gherardi, Hölzl, Nobrega and Pauly [9].

► **Theorem 5** (Positive  $G_\delta$ -Choice).  $\Pi_2^0 PC_{2^{\mathbb{N}}} \equiv_{sW} PC'_{\mathbb{R}}$ .

In our context  $PC'_{\mathbb{R}}$  is somewhat easier to handle than  $\Pi_2^0 PC_{2^{\mathbb{N}}}$  and Theorems 4 and 5 lead to the following corollary.

► **Corollary 6** (Monte Carlo computability).  $f \leq_W PC'_{\mathbb{R}}$  if and only if  $f$  is Monte Carlo computable.

Bienvenu and Kuyper [1] answered a number of questions related to composition and they proved the following result on the compositional product of  $PC'_{\mathbb{R}}$ .

► **Theorem 7** (Composition).  $PC'_{\mathbb{R}} * PC'_{\mathbb{R}} \equiv_W PC'_{\mathbb{R}}$ .

In light of Theorem 4 and Theorem 5 we obtain a second independent proof of this theorem along the lines of the Independent Choice Theorem of Brattka, Gherardi and Hölzl [8, Theorem 4.3], which is essentially based on Fubini's Theorem. The synthetic representation of  $\Pi_2^0(2^{\mathbb{N}})$  allows us to transfer the proof of the Independent Choice Theorem directly to a proof of the fact that  $\Pi_2^0 PC_{2^{\mathbb{N}}} * \Pi_2^0 PC_{2^{\mathbb{N}}} \equiv_W \Pi_2^0 PC_{2^{\mathbb{N}}}$  and hence we obtain Theorem 7 with the help of Theorem 5. The importance of Theorem 7 for us lies in the following conclusion.

► **Corollary 8.** *Monte Carlo computable functions are closed under composition.*

This conclusion ensures that Monte Carlo computability satisfies one of the necessary conditions that any reasonable concept of computability should satisfy. Due to the definition it is also clear that every Las Vegas computable function is Monte Carlo computable.

► **Corollary 9.** *Every Las Vegas computable function is Monte Carlo computable.*



The inverse implication is clearly false, as there are functions that are Monte Carlo computable but not Las Vegas computable. An example is the equality test  $=: \mathbb{R} \times \mathbb{R} \rightarrow \{0, 1\}$ ; it is Monte Carlo computable since it is reducible to  $C_{\mathbb{N}} \leq_W PC_{\mathbb{R}}$ , but cannot be Las Vegas computable since it is not even non-deterministically computable [6].

#### 4 Weak Weak König's Lemma and Jumps

In this section we discuss the relation of Weak Weak König's Lemma to Monte Carlo computations. Weak König's Lemma and Weak Weak König's Lemma are principles that have been intensively studied in reverse mathematics [26]. The classical lemma of König says (in its weak version) that every infinite binary tree has an infinite path. Here we understand Weak König's Lemma as the mathematical problem

$$\text{WKL}: \subseteq \text{Tr} \Rightarrow 2^{\mathbb{N}}, T \mapsto [T]$$

that maps an infinite binary tree  $T \subseteq 2^*$  to an infinite path  $p \in [T]$  of this tree. By  $\text{Tr}$  we denote the set of all binary trees (represented via their characteristic functions) and by  $[T]$  we denote the set of infinite paths of such a tree. We assume that  $\text{dom}(\text{WKL})$  is the set of infinite binary trees. Weak Weak König's Lemma is the restriction of WKL to trees  $T$  such that  $\mu_{2^{\mathbb{N}}}([T]) > 0$ , that is, such that the set of infinite paths has positive measure. Some basic facts known about these principles are the following (see [21, 8, 10]):

- $\text{WKL} \equiv_{sW} C_{2^{\mathbb{N}}}$ .
- $\text{WWKL} \equiv_{sW} PC_{2^{\mathbb{N}}}$ .
- $\text{WWKL} \times C_{\mathbb{N}} \equiv_{sW} PC_{\mathbb{R}}$ .
- $\text{WWKL} <_{sW} \text{WKL}$  and  $\text{WWKL} <_{sW} PC_{\mathbb{R}}$ .
- $K_{\mathbb{N}} <_{sW} \text{WWKL}$  and  $C_{\mathbb{N}} <_{sW} PC_{\mathbb{R}}$ .

Here  $K_{\mathbb{N}} := C_2^*$  is also called *compact choice* because it can be seen as  $C_{\mathbb{N}}$  restricted to sets  $A \subseteq \mathbb{N}$  that are given together with an upper bound.

In the context of Monte Carlo computability we need to transfer some of these results to jumps of the involved principles. For the forwards direction one can usually use monotonicity of jumps with respect to strong Weihrauch reducibility. The following lemma was proved by Brattka, Gherardi and Marcone [11, Proposition 5.6].

► **Lemma 10** (Monotonicity).  $f \leq_{sW} g \implies f' \leq_{sW} g'$ .

We note that a corresponding result for ordinary Weihrauch reducibility  $\leq_W$  does not hold. This is one of the reasons why the study of strong Weihrauch reductions seems to be unavoidable. Even if one is interested only in results about ordinary Weihrauch reducibility of jumps, this typically requires to study strong Weihrauch reductions.

Surprisingly, we were also able to prove a certain inverse result of the above monotonicity property, which we formulate next. Here “relative to the halting problem” is supposed to mean that the reduction functions  $H, K$  used for the reduction  $f \leq_W g$  both have access to the halting problem.

► **Theorem 11** (Jumps and relativization).  $f' \leq_W g' \implies f \leq_W g$  relative to the halting problem.

**Proof.** Let  $f' \leq_W g'$ . Then there are computable functions  $H, K: \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow \mathbb{N}^{\mathbb{N}}$  such that  $H \langle r, GK(r) \rangle$  is a name for an output of  $f'$  on the input specified by  $r \in \mathbb{N}^{\mathbb{N}}$ , whenever  $G$  is a realizer for  $g'$ . For  $r$  actually any sequence is allowed that converges to an input  $q$  of  $f$  and  $K(r)$  must be a sequence converging to a name of an input of  $g$  in this situation. Without loss of generality we can assume that  $\text{range}(K) \subseteq 2^{\mathbb{N}}$ . By a theorem of Brattka, Hendtlass

and Kreuzer [12, Theorem 14.11] there are functions  $K_0: \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow 2^{\mathbb{N}}$  and  $K_1: \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow \mathbb{N}^{\mathbb{N}}$  that are computable with access to the halting problem and such that

$$K_0(q) = \lim K K_1(q) \text{ and } \lim K_1(q) = q$$

for all  $q \in \text{dom}(\lim \circ K \circ \lim^{-1})$ . Then  $K_1(q) = r$  is a sequence that converges to  $q$  and such that  $K_0(q) = \lim K(r)$  is a name of an input of  $g$ . Let  $H_0$  be defined by  $H_0\langle q, u \rangle := H\langle K_1(q), u \rangle$  for all  $q, u$ . Then  $H_0$  is computable relative to the halting problem and we obtain that  $G \lim$  is a realizer of  $g'$  for every realizer  $G$  of  $g$  and hence

$$H_0\langle q, G K_0(q) \rangle = H\langle K_1(q), G K_0(q) \rangle = H\langle r, G \lim K(r) \rangle,$$

which is name for an output of  $f'$  on input  $r$  and hence an output for  $f$  on input  $\lim r = q$ . This proves  $f \leq_W g$  relative to the halting problem. ◀

An analogous statement holds for  $\leq_{sW}$  in place of  $\leq_W$ . Often separations of problems are proved in a topological way by showing that there are not even continuous reduction functions  $H, K$ . In such a situation,  $H, K$  cannot even be computable with respect to the halting problem. Hence, topological separations of  $f$  and  $g$  are very useful when it comes to separating  $f'$  and  $g'$ .

The diagram in the upper half of Figure 5 illustrates some important reductions for jumps of Weak Weak König's Lemma and related principles

## 5 Sorting

Sorting infinite sequences is a basic computational task that was introduced and studied by Neumann and Pauly [22] in the binary case. We generalize this problem by defining  $\text{SORT}_n: \{0, 1, \dots, n-1\}^{\mathbb{N}} \rightarrow \{0, 1, \dots, n-1\}^{\mathbb{N}}$  by

$$\text{SORT}_n(p) := 0^{k_0} 1^{k_1} \dots (m-1)^{k_{m-1}} \widehat{m}$$

if  $m < n$  is the smallest digit that appears infinitely often in  $p$  and each digit  $i < m$  appears exactly  $k_i$  times in  $p$ . Here  $\widehat{m} = mmm\dots$  denotes the infinite sequence which has the constant value  $m$ . This definition is understood such that  $\text{SORT}_n(p) = \widehat{0}$  if 0 appears infinitely often in  $p$ . In Figure 4  $\text{SORT}_5$  is illustrated.

For  $n \leq 1$  the problem  $\text{SORT}_n$  is computable, since  $\text{SORT}_0$  is the nowhere defined function and  $\text{SORT}_1$  is the constant function. Neumann and Pauly [22, Proposition] proved that  $C_{\mathbb{N}} \leq_W \text{SORT}_2$  and in fact it is easy to see that even a strong reduction holds.

► **Proposition 12.**  $C_{\mathbb{N}} \leq_{sW} \text{SORT}_2$ .

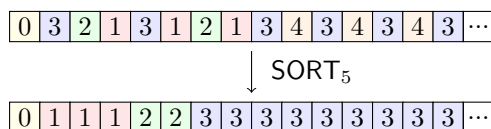
Here we want to discuss probabilistic solutions for the problem  $\text{SORT}_n$  and the interesting observation is the following.

► **Proposition 13.**  $\text{SORT}_n \leq_{sW} \text{WWKL}'$  for all  $n \in \mathbb{N}$ .

**Proof.** Given a sequence  $p \in \{0, \dots, n-1\}^{\mathbb{N}}$  as input to  $\text{SORT}_n$  we want to compute  $\text{SORT}_n(p)$  with the help of  $\text{WWKL}'$ . We produce a sequence of binary trees  $(T_i)_i$  with  $T_i := \bigcup_{m=0}^{n-1} T_{i,m}$ . Here for each  $m \in \{0, \dots, n-1\}$

$$T_{i,m} := 0^m 10^{k_0} 10^{k_1} \dots 10^{k_{m-1}} 1\{0, 1\}^{k_m},$$

where  $k_j$  denotes the number of appearances of digit  $j \in \{0, \dots, m\}$  in the initial segment  $p(0), \dots, p(i)$ . The sequence  $(T_{i,m})_i$  converges to a tree  $S_m$ . If no  $j \leq m$  appears infinitely



■ **Figure 4** Application of SORT<sub>5</sub> to some example sequence (that continues alternately with 4, 3).

often in  $p$ , then  $[S_m] = \emptyset$ . If  $m$  appears infinitely often in  $p$  and no smaller  $j < m$  appears infinitely often in  $p$ , then

$$[S_m] = 0^m 10^{k_0} 10^{k_1} \dots 10^{k_{m-1}} 1 \{0, 1\}^{\mathbb{N}},$$

where  $k_j$  is the number of appearances of digit  $j$  in  $p$  for  $j \in \{0, \dots, m-1\}$ . If  $l < m$  is the minimal digit that appears infinitely often in  $p$ , then

$$[S_m] = \{0^m 10^{k_0} 10^{k_1} \dots 10^{k_{l-1}} \widehat{10}\},$$

where  $k_j$  is the number of appearances of digit  $j$  in  $p$  for  $j \in \{0, \dots, l-1\}$ . The sequence  $(T_i)_i$  converges to the tree  $T := \bigcup_{m=0}^{n-1} S_m$ . Since one digit  $m < n$  is the minimal digit that appears infinitely often in  $p$ , we have  $\mu([T]) \geq \mu([S_m]) > 0$ . Given some  $q \in [T]$ , we can reconstruct SORT <sub>$n$</sub> ( $p$ ), since there is some  $m < n$  and there is some  $l < m$  or some  $r \in \{0, 1\}^{\mathbb{N}}$  such that exactly one of the following cases holds:

1.  $q = 0^m 10^{k_0} 10^{k_1} \dots 10^{k_{l-1}} \widehat{10} \implies \text{SORT}_n(p) = 0^{k_0} 1^{k_1} \dots (l-1)^{k_{l-1}} \widehat{l}$ ,
2.  $q = 0^m 10^{k_0} 10^{k_1} \dots 10^{k_{m-1}} 1r \implies \text{SORT}_n(p) = 0^{k_0} 1^{k_1} \dots (m-1)^{k_{m-1}} \widehat{m}$ . ◀

Brattka, Gherardi and Hölzl proved that WWKL (and hence WWKL') is strongly idempotent [8, Corollary 4.5], that is,  $\text{WWKL}' \times \text{WWKL}' \equiv_{\text{sW}} \text{WWKL}'$ , and hence Proposition 13 can be strengthened in the following way.

▶ **Corollary 14.** SORT <sub>$n$</sub> <sup>\*</sup>  $\leq_{\text{sW}}$  WWKL' for all  $n \in \mathbb{N}$ .

Since  $\text{WWKL}' \leq_{\text{W}} \text{PC}'_{\mathbb{R}}$  we also obtain the following conclusion.

▶ **Corollary 15.** SORT <sub>$n$</sub> <sup>\*</sup> is Monte Carlo computable for every  $n \in \mathbb{N}$ .

The next observation is that the Intermediate Value Theorem IVT is reducible to SORT<sub>2</sub>. We study IVT here in the slightly easier form of connected choice CC<sub>[0,1]</sub>, that is, C<sub>[0,1]</sub> restricted to connected subsets of  $[0, 1]$ . The equivalence  $\text{IVT} \equiv_{\text{sW}} \text{CC}_{[0,1]}$  was proved by Brattka and Gherardi [5, Theorem 6.2] (where CC<sub>[0,1]</sub> appears under the name C<sub>I</sub>).

▶ **Proposition 16.** CC<sub>[0,1]</sub>  $\leq_{\text{W}}$  SORT<sub>2</sub>.

**Proof.** Let two monotone rational sequences  $(a_n)_n$  and  $(b_n)_n$  in  $[0, 1]$  be given, the first one increasing, the second one decreasing, and such that  $a_n \leq b_n$  for all  $n$ . We computably produce an input  $p$  for SORT<sub>2</sub> by writing a 1 every second step and by producing occasionally a 0 according to the following algorithm: whenever we find an  $n \in \mathbb{N}$  such that  $|b_n - a_n| < 2^{-k}$ , then we ensure that we have  $k$  digits 0 included in  $p$ . Hence,  $p$  will include exactly  $k$  zeros if  $k$  is the largest number such that there is some  $n$  with  $|b_n - a_n| < 2^{-k}$  and it will include infinitely many zeros if for every  $k$  there is such an  $n$ . Given the output of SORT<sub>2</sub>( $p$ ) and the original input  $(a_n)_n$  and  $(b_n)_n$  we can find a point  $x \in [0, 1]$  with  $a_n \leq x \leq b_n$  for all  $n$  as follows. If we see at least  $k$  zeros in SORT<sub>2</sub>( $p$ ), then we search for  $n$  with  $|b_n - a_n| < 2^{-k}$  and produce  $x_k := a_n + \frac{1}{2}(b_n - a_n)$  as approximation of  $x$  of precision  $2^{-k}$ . In the moment where

## 17:10 Monte Carlo Computability

we see that there are  $k$  and not more zeros in  $\text{SORT}_2(p)$ , that is, we see the first 1 at position  $k + 1$ , we continue to produce  $x_{k+i} := x_k$  as approximation for  $x$  of any higher precision  $2^{-k-i}$ . Since there is no  $n$  with  $|b_n - a_n| < 2^{-k-1}$ , it is guaranteed that  $a_n \leq x_{k+i} \leq b_n$  for all  $i \in \mathbb{N}$ .  $\blacktriangleleft$

In particular, this result implies that zero finding, that is, the Intermediate Value Theorem IVT, is Monte Carlo computable.

► **Corollary 17.** *The Intermediate Value Theorem IVT is Monte Carlo computable.*

Brattka, Gherardi and Hölzl proved  $\text{CC}_{[0,1]} \leq_W \text{WWKL}'$  [8, Corollary 15.9], which also implies Corollary 17. However, Proposition 16 generalizes the aforementioned result via Proposition 13. We note that  $\text{CC}_{[0,1]} \not\leq_{sW} \text{SORT}_2$  since  $\text{IVT} \not\leq_{sW} \text{WWKL}'$  by [8, Corollary 15.8]. Brattka and Rakotoniaina proved  $\text{CC}_{[0,1]} \leq_W C'_\mathbb{N}$  [17, Proposition 5.21], a result which is generalized by the following observation.

► **Proposition 18.**  *$\text{SORT}_n \leq_{sW} C'_\mathbb{N}$  for all  $n \in \mathbb{N}$ .*

**Proof.** By [11, Theorem 9.4] we have  $\text{CL}_\mathbb{N} \equiv_{sW} C'_\mathbb{N}$ , where  $\text{CL}_\mathbb{N}$  denotes the cluster point problem (i.e., the problem to find a cluster point of a given sequence of natural numbers that has one). Given a sequence  $p \in \{0, 1, \dots, n-1\}^\mathbb{N}$  we compute a sequence  $q \in \mathbb{N}^\mathbb{N}$  as follows: we inspect  $p$  and whenever we find a new occurrence of the digit  $m \in \{0, 1, \dots, n-1\}$ , then we add a number  $\langle m, \langle k_0, k_1, \dots, k_{m-1} \rangle \rangle \in \mathbb{N}$  to  $q$ , where each  $k_i \in \mathbb{N}$  counts the occurrences of digit  $i$  in  $p$  so far. One fixed such tuple  $\langle m, \langle k_0, k_1, \dots, k_{m-1} \rangle \rangle$  appears infinitely often in  $q$  if and only if  $m$  appears infinitely often in  $p$  and each number  $i \in \{0, 1, \dots, m-1\}$  appears exactly  $k_i$  times in  $p$  altogether.  $\text{CL}_\mathbb{N}(q)$  determines one such tuple that appears infinitely often and hence it is easy to reconstruct  $\text{SORT}_n(p)$  from  $\text{CL}_\mathbb{N}(q)$ .  $\blacktriangleleft$

We note that the proof even shows  $\text{SORT}_n \leq_{sW} \text{UC}'_\mathbb{N}$ , where  $\text{UC}'_\mathbb{N}$  is the unique version of  $C'_\mathbb{N}$ . Proposition 18 also yields an independent proof of Corollary 15. With the help of the fact that  $C'_\mathbb{N} \times C'_\mathbb{N} \equiv_{sW} C'_\mathbb{N}$  we obtain the following corollary.

► **Corollary 19.**  *$\text{SORT}_n^* \leq_{sW} C'_\mathbb{N}$  for all  $n \in \mathbb{N}$ .*

One could ask whether this result can be strengthened to the upper bound  $K'_\mathbb{N}$ , which is not the case.

► **Proposition 20.**  *$\text{CC}_{[0,1]} \not\leq_W K'_\mathbb{N}$ .*

**Proof.** By [11, Theorem 9.4] we have  $K'_\mathbb{N} \equiv_{sW} \text{BWT}_\mathbb{N}$ , where  $\text{BWT}_\mathbb{N}$  denotes the restriction of  $\text{CL}_\mathbb{N}$  to bounded sequences. Let us assume that  $\text{CC}_{[0,1]} \leq_W \text{BWT}_\mathbb{N}$  holds via computable reduction functions  $K, H$ . We construct sequences  $(a_n)_n$  and  $(b_n)_n$  of rational numbers in  $[0, 1]$  with  $a_n \leq a_{n+1} \leq b_{n+1} \leq b_n$  for all  $n \in \mathbb{N}$  as an input for  $\text{CC}_{[0,1]}$ . We start with choosing  $a_n := 0$  and  $b_n := 1$  for all  $n \in \mathbb{N}$ . Upon this input  $K$  will produce a sequence  $(x_n)_n$  of natural numbers as an input to  $\text{BWT}_\mathbb{N}$ . This sequence has to contain only finitely many different numbers  $y_0, \dots, y_j$ . Each of these numbers is a possible output of  $\text{BWT}_\mathbb{N}$  and hence for each of these numbers  $y_i$  and the original input,  $H$  has to select a point  $z_i \in [0, 1]$  with  $a_n \leq z_i \leq b_n$  for all  $n \in \mathbb{N}$ . Since  $H$  is continuous, there is a certain input length  $k$  such that upon input of  $a_0, \dots, a_k$  and  $b_0, \dots, b_k$  the function  $H$  already approximates each number  $z_i$  up to precision  $2^{-j-2}$ . It is clear that  $j+1$  intervals of diameter  $2^{-j-1}$  cannot cover the entire interval  $[0, 1]$  and in fact the uncovered part has non-empty interior. We can also assume that  $k$  is large enough such that by continuity of  $K$  actually all the numbers  $y_0, \dots, y_j$  already appear in the output of  $K$  upon input of  $a_0, \dots, a_k$  and  $b_0, \dots, b_k$ . Now we

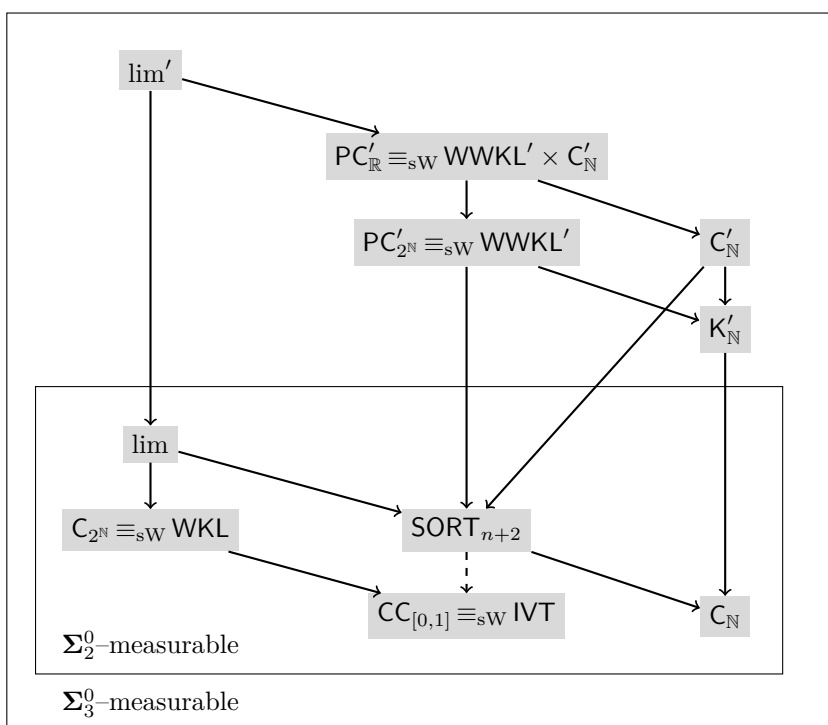


Figure 5 Sorting in the Weihrauch lattice.

can choose a new proper interval  $[a, b]$  that does not overlap with any of the approximations of the  $z_i$  and we modify the original input by choosing  $a_{k+i} := a$  and  $b_{k+i} := b$  for all  $i \in \mathbb{N}$ . Upon this modified input  $K$  has to produce a new number  $y \in \mathbb{N}$ , which is not among the numbers  $y_0, \dots, y_j$ , since otherwise the output that is produced by the reduction is not in  $[a, b]$ . We can now inductively repeat the above construction and in this way we construct an input  $(a_n)_n, (b_n)_n$  to  $CC_{[0,1]}$  such that  $K$  produces a sequence  $(x_n)_n$  with infinitely many different numbers  $x_n$  in its range (none of which appear infinitely often, in fact). This is not an admissible input to  $BWT_{\mathbb{N}}$  and hence a contradiction. ◀

Neumann and Pauly [22, Corollary 27] proved that  $SORT_2$  is low<sub>2</sub> in the sense that  $\lim * \lim * SORT_2 \equiv_W \lim * \lim$ . However, it is not closed under composition and not low as, for instance, the following result shows.

► **Proposition 21.**  $C'_n \leq_W SORT_n * SORT_n \leq_W C'_N$  for all  $n \in \mathbb{N}$ .

**Proof.** We obtain  $C'_n \leq_W \lim_n * SORT_n \leq_W C_N * SORT_n \leq_W SORT_n * SORT_n \leq_W C'_N$ . The first reduction holds since  $SORT_n(p)$  is always a converging sequence and its limit is a cluster point of  $p$ . We have  $C'_n \equiv_W CL_n$  for the cluster point problem  $CL_n$  by a result of Brattka, Gherardi and Marcone [11, Theorem 9.4]. The second reduction follows since  $\lim_n \leq_W \lim_N \leq_W C_N$  and the third reduction follows from Proposition 12. The reduction  $SORT_n * SORT_n \leq_W C'_N$  follows from Proposition 18 and the fact that  $C'_N * C'_N \equiv_W C'_N$  (which one can easily see by a direct proof). ◀

As a consequence of this result we obtain  $SORT_n \not\leq_W WKL$  for  $n \geq 2$  since  $WKL * WKL \equiv_W WKL$  by [21, Theorem 6.14] and hence we get the following conclusion.

► **Corollary 22.**  $SORT_n$  is not non-deterministically computable for  $n \geq 2$ .

In the diagram in Figure 5 we have collected some of the results on sorting. The solid lines indicate strong Weihrauch reductions against the direction of the arrows, the dashed line indicates an ordinary Weihrauch reduction against the direction of the arrow.

## 6 Algebraic Computation Models

One reason that Neumann and Pauly studied the binary sorting problem is that  $\text{SORT}_2^*$  characterizes the strength of strongly analytic machines as defined by Gärtner and Hotz [19, 20]. Essentially these strongly analytic machines are real random access machines over the field of real numbers (with computable constants) as studied by Blum, Shub and Smale and others [3] and extended by semantics that allows one to approximate the output. They are called *strongly analytic* if the machine also provides an error bound for the approximation of the output. Neumann and Pauly proved the following theorem [22, Observation 30, Corollary 34 and 11].

► **Theorem 23** (Algebraic machine models). *Consider a function of type  $f: \mathbb{R}^* \rightarrow \mathbb{R}^*$ .*

1. *If  $f$  is computable by a strongly analytic machine, then  $f \leq_W \text{SORT}_2^*$  and  $\text{SORT}_2^*$  is equivalent to a function computable by a strongly analytic machine.*
2. *If  $f$  is computable by a BSS machine, then  $f \leq_W C_{\mathbb{N}}$  and  $C_{\mathbb{N}}$  is equivalent to a function computable by a BSS machine.*

Using this result we obtain the following corollary; note that while the class of functions computable by strongly analytic machines is not closed under composition, the corollary *does* hold for such compositions as well.

► **Corollary 24.** *Any finite composition of functions  $f: \mathbb{R}^* \rightarrow \mathbb{R}^*$  that are computable on strongly analytic machines is Monte Carlo computable.*

Since functions that are computable on a Blum, Shub and Smale machine (BSS machine) are a special case, we obtain the following corollary.

► **Corollary 25.** *Every function  $f: \mathbb{R}^* \rightarrow \mathbb{R}^*$  that is computable on a BSS machine is Monte Carlo computable.*

Corollary 25 could already be deduced from the observation that  $C_{\mathbb{N}} \leq_W \text{PC}_{\mathbb{R}}$  and does not require our results on sorting. However, it is worth pointing out that in general functions computable on a BSS machine (even the equality test) are not non-deterministically computable and, in particular, not Las Vegas computable.

---

## References

- 1 Laurent Bienvenu and Rutger Kuyper. Parallel and serial jumps of Weak König's Lemma. In Adam Day, Michael Fellows, Noam Greenberg, Bakhadyr Khoussainov, Alexander Melnikov, and Frances Rosamond, editors, *Computability and Complexity: Essays Dedicated to Rodney G. Downey on the Occasion of His 60th Birthday*, volume 10010 of *Lecture Notes in Computer Science*, pages 201–217. Springer, Cham, 2017. doi: 10.1007/978-3-319-50062-1\_15.
- 2 Laurent Bienvenu and Christopher P. Porter. Deep  $\Pi_1^0$  classes. *Bulletin of Symbolic Logic*, 22(2):249–286, 2016. doi: 10.1017/bsl.2016.9.
- 3 Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and Real Computation*. Springer, New York, 1998.

- 4 Vasco Brattka, Matthew de Brecht, and Arno Pauly. Closed choice and a uniform low basis theorem. *Annals of Pure and Applied Logic*, 163:986–1008, 2012. doi:10.1016/j.apal.2011.12.020.
- 5 Vasco Brattka and Guido Gherardi. Effective choice and boundedness principles in computable analysis. *The Bulletin of Symbolic Logic*, 17(1):73–117, 2011. doi:10.2178/bsl/1294186663.
- 6 Vasco Brattka and Guido Gherardi. Weihrauch degrees, omniscience principles and weak computability. *The Journal of Symbolic Logic*, 76(1):143–176, 2011. doi:10.2178/jsl/1294170993.
- 7 Vasco Brattka, Guido Gherardi, and Rupert Hölzl. Las Vegas computability and algorithmic randomness. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 130–142, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2015.130.
- 8 Vasco Brattka, Guido Gherardi, and Rupert Hölzl. Probabilistic computability and choice. *Information and Computation*, 242:249–286, 2015. doi:10.1016/j.ic.2015.03.005.
- 9 Vasco Brattka, Guido Gherardi, Rupert Hölzl, Hugo Nobrega, and Arno Pauly. Positive Borel choice. Unpublished draft, 2016.
- 10 Vasco Brattka, Guido Gherardi, Rupert Hölzl, and Arno Pauly. The Vitali covering theorem in the Weihrauch lattice. In Adam Day, Michael Fellows, Noam Greenberg, Bakhadyr Khoussainov, Alexander Melnikov, and Frances Rosamond, editors, *Computability and Complexity: Essays Dedicated to Rodney G. Downey on the Occasion of His 60th Birthday*, volume 10010 of *Lecture Notes in Computer Science*, pages 188–200. Springer, Cham, 2017. doi:10.1007/978-3-319-50062-1\_14.
- 11 Vasco Brattka, Guido Gherardi, and Alberto Marcone. The Bolzano-Weierstrass theorem is the jump of weak König’s lemma. *Annals of Pure and Applied Logic*, 163:623–655, 2012. doi:10.1016/j.apal.2011.10.006.
- 12 Vasco Brattka, Matthew Hendtlass, and Alexander P. Kreuzer. On the uniform computational content of computability theory. arXiv 1501.00433, 2015. arXiv:1501.00433.
- 13 Vasco Brattka, Peter Hertling, and Klaus Weihrauch. A tutorial on computable analysis. In S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *New Computational Paradigms: Changing Conceptions of What is Computable*, pages 425–491. Springer, New York, 2008. doi:10.1007/978-0-387-68546-5\_18.
- 14 Vasco Brattka, Stéphane Le Roux, Joseph S. Miller, and Arno Pauly. The Brouwer fixed point theorem revisited. In Arnold Beckmann, Laurent Bienvenu, and Nataša Jonoska, editors, *Pursuit of the Universal*, volume 9709 of *Lecture Notes in Computer Science*, pages 58–67, Switzerland, 2016. Springer. 12th Conference on Computability in Europe, CiE 2016, Paris, France, June 27 - July 1, 2016. doi:10.1007/978-3-319-40189-8\_6.
- 15 Vasco Brattka and Arno Pauly. Computation with advice. In Xizhong Zheng and Ning Zhong, editors, *CCA 2010, Proceedings of the Seventh International Conference on Computability and Complexity in Analysis*, Electronic Proceedings in Theoretical Computer Science, pages 41–55, 2010. doi:10.4204/EPTCS.24.9.
- 16 Vasco Brattka and Arno Pauly. On the algebraic structure of Weihrauch degrees. arXiv 1604.08348, 2016. arXiv:1604.08348.
- 17 Vasco Brattka and Tahina Rakotoniaina. On the uniform computational content of Ramsey’s theorem. arXiv 1508.00471, 2015. arXiv:1508.00471.
- 18 François G. Dorais, Damir D. Dzharfarov, Jeffrey L. Hirst, Joseph R. Mileti, and Paul Shafer. On uniform relationships between combinatorial problems. *Transactions of the American Mathematical Society*, 368(2):1321–1359, 2016. doi:10.1090/tran/6465.

- 19 Tobias Gärtner and Günter Hotz. Computability of analytic functions with analytic machines. In *Mathematical theory and computational practice*, volume 5635 of *Lecture Notes in Comput. Science*, pages 250–259. Springer, Berlin, 2009. doi:10.1007/978-3-642-03073-4\_26.
- 20 Tobias Gärtner and Günter Hotz. Representation theorems for analytic machines and computability of analytic functions. *Theory of Computing Systems*, 51(1):65–84, 2012. doi:10.1007/s00224-011-9374-z.
- 21 Guido Gherardi and Alberto Marcone. How incomputable is the separable Hahn-Banach theorem? *Notre Dame Journal of Formal Logic*, 50(4):393–425, 2009. doi:10.1215/00294527-2009-018.
- 22 Eike Neumann and Arno Pauly. A topological view on algebraic computation models. arXiv 1602.08004, 2016. <http://arxiv.org/abs/1602.08004>.
- 23 Arno Pauly. How incomputable is finding Nash equilibria? *Journal of Universal Computer Science*, 16(18):2686–2710, 2010. doi:10.3217/jucs-016-18-2686.
- 24 Arno Pauly. On the (semi)lattices induced by continuous reducibilities. *Mathematical Logic Quarterly*, 56(5):488–502, 2010. doi:10.1002/malq.200910104.
- 25 Arno Pauly. The descriptive theory of represented spaces. arXiv 1408.5329, 2014. <http://arxiv.org/abs/1408.5329>.
- 26 Stephen G. Simpson. *Subsystems of Second Order Arithmetic*. Perspectives in Logic, Association for Symbolic Logic. Cambridge University Press, Poughkeepsie, second edition, 2009.
- 27 Thorsten von Stein. *Vergleich nicht konstruktiv lösbarer Probleme in der Analysis*. Fachbereich Informatik, FernUniversität Hagen, 1989. Diplomarbeit.
- 28 Klaus Weihrauch. *Computable Analysis*. Springer, Berlin, 2000.
- 29 Martin Ziegler. Real hypercomputation and continuity. *Theory of Computing Systems*, 41(1):177–206, 2007. doi:10.1007/s00224-006-1343-6.



# The Parameterized Complexity of Finding a 2-Sphere in a Simplicial Complex\*

Benjamin Burton<sup>1</sup>, Sergio Cabello<sup>2</sup>, Stefan Kratsch<sup>3</sup>, and William Pettersson<sup>4</sup>

1 School of Mathematics and Physics, The University of Queensland, Brisbane, Australia

`bab@maths.uq.edu.au`

2 Department of Mathematics, IMFM, and Department of Mathematics, FMF, University of Ljubljana, Ljubljana, Slovenia

`sergio.cabello@fmf.uni-lj.si`

3 Institute of Computer Science, University of Bonn, Bonn, Germany

`kratsch@cs.uni-bonn.de`

4 School of Science, RMIT University, Melbourne, Australia

`william@ewpettersson.se`

---

## Abstract

We consider the problem of finding a subcomplex  $\mathcal{K}'$  of a simplicial complex  $\mathcal{K}$  such that  $\mathcal{K}'$  is homeomorphic to the 2-dimensional sphere,  $\mathbb{S}^2$ . We study two variants of this problem. The first asks if there exists such a  $\mathcal{K}'$  with at most  $k$  triangles, and we show that this variant is  $W[1]$ -hard and, assuming ETH, admits no  $\mathcal{O}(n^{o(\sqrt{k})})$  time algorithm. We also give an algorithm that is tight with regards to this lower bound. The second problem is the dual of the first, and asks if  $\mathcal{K}'$  can be found by removing at most  $k$  triangles from  $\mathcal{K}$ . This variant has an immediate  $\mathcal{O}(3^k \text{poly}(|\mathcal{K}|))$  time algorithm, and we show that it admits a polynomial kernelization to  $\mathcal{O}(k^2)$  triangles, as well as a polynomial compression to a weighted version with bit-size  $\mathcal{O}(k \log k)$ .

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** computational topology, parameterized complexity, simplicial complex

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.18

## 1 Introduction

Topology is the study of the properties of spaces that are preserved under continuous deformations of the space. Intuitively, this can be summed up by the joke description of a topologist as a mathematician who cannot tell the difference between a coffee mug and a doughnut, as each can be continuously deformed into the other. In this paper we discuss manifolds, which are topological spaces that locally look like Euclidean space. That is to say, every point in a  $d$ -manifold (without boundary) has a neighborhood homeomorphic to  $\mathbb{R}^d$ .

The simplest manifold is the  $d$ -sphere, which is the boundary of a  $(d+1)$ -dimensional ball, where the  $(d+1)$ -dimensional ball is simply a closed neighborhood of  $\mathbb{R}^{d+1}$ . In particular, the 2-sphere which we will discuss is the 2-dimensional surface of a 3-dimensional ball (such as a soccer ball) that would live in the 3-space of our physical world. The sphere is of interest as it relates to the *connected sum* operation on manifolds. A connected sum of two  $d$ -manifolds is found by removing a  $(d+1)$ -dimensional ball from each manifold, and identifying the two

---

\* Partially supported by the Slovenian Research Agency, program P1-0297 and project L7-5459.



components along the boundaries of the respective balls. The  $d$ -sphere forms the identity element of this operation. Finding embedded  $d$ -spheres that can separate a manifold into two non-trivial components is therefore the topological equivalent of the factorization of integers. Indeed, a manifold that has no such spheres is called *prime*, and a *prime decomposition* of a manifold is a decomposition into prime manifolds.

In this paper we will use (*abstract*) *simplicial complexes* to combinatorially represent manifolds. At an informal level, a simplicial complex is a collection of simplices that are glued by identifying some faces. In principle, the abstract simplicial complex does not live in any ambient space, although we can always represent it geometrically using spaces of high enough dimension. A formal definition is given in Section 2.

Arguably, the most natural question to ask regarding a simplicial complex is whether it represents a manifold. The question is easy to answer for 2-manifolds: it suffices to check whether each edge is adjacent to exactly two triangles. Additionally, in two dimensions we can recognize the manifold by calculating the Euler characteristic of the simplicial complex, itself a simple enumeration of vertices, edges and faces, and checking whether it is orientable. Recognizing the manifold of a simplicial 3-complex is far harder, even for the 3-sphere [19]. The recognition of 4-dimensional manifolds and the 5-sphere is an undecidable problem (see for example the appendix of [18]), while the recognition of the 4-sphere is a notorious open problem. Interestingly, in dimensions 4 and higher there exists manifolds (such as the  $E_8$  manifold) which can not even be represented as a simplicial complex [10].

**Our work.** We return to a basic problem for 2-dimensional simplicial complexes: does a given simplicial complex contain a subcomplex that is (homeomorphic to) a 2-sphere? The problem is known to be NP-hard, and we study its parameterized complexity with respect to the solution size (number of triangles in the subcomplex) and its dual (number of triangles not in the subcomplex); we begin with the former problem.

2-DIM-SPHERE

**Input:** A pair  $(\mathcal{K}, k)$  where  $\mathcal{K}$  is a 2-dimensional simplicial complex and  $k$  is a positive integer.

**Question:** Does  $\mathcal{K}$  contain a subcomplex with at most  $k$  triangles that is homeomorphic to the 2-dimensional sphere?

We show that this problem is  $W[1]$ -hard with respect to  $k$ . In fact we show that, assuming the Exponential Time Hypothesis (ETH; see preliminaries), the problem cannot be solved in  $n^{o(\sqrt{k})}$  time. ETH implies a core hypothesis of parameterized complexity, namely that  $FPT \neq W[1]$  (comparable to the hypothesis that  $P \neq NP$ ). Together with its twin SETH (the Strong Exponential Time Hypothesis) it is known to imply a wide range of lower bounds, often matching known algorithmic results, for various NP-hard problems. (To note, a very active branch of research uses SETH for tight lower bounds for problems in P.)

► **Theorem 1.** *The 2-DIM-SPHERE problem is  $W[1]$ -hard with respect to parameter  $k$  and, unless ETH fails, it has no  $f(k)n^{o(\sqrt{k})}$ -time algorithm for any computable function  $f$ .*

Note that the related problem variant of finding a subcomplex with *at least*  $k$  triangles that is homeomorphic to the 2-dimensional sphere is NP-hard for  $k = 0$ , as this is simply NP-hard problem of testing whether there is any subcomplex that is homeomorphic to the 2-sphere. (Note that hardness for finding a subcomplex with at most  $k$  triangles also implies hardness for finding one with *exactly*  $k$  triangles.)

We complement Theorem 1 by giving an algorithm for 2-DIM-SPHERE that runs in  $n^{O(\sqrt{k})}$  time, which is essentially tight; it can also be used to find a solution with exactly  $k$  triangles.

► **Theorem 2.** *The 2-DIM-SPHERE problem can be solved in time  $n^{\mathcal{O}(\sqrt{k})}$ .*

For the dual problem, we are interested in the parameterized complexity relative to the number  $k$  of triangles that are not in the solution (i.e., not in the returned subcomplex that is homeomorphic to the 2-sphere). In other words, the question becomes that of deleting  $k$  triangles (plus edges and vertices that are only incident with these triangles) to obtain a subcomplex that is homeomorphic with the 2-sphere. Similarly to before, deleting *at least*  $k$  triangles is NP-hard for  $k = 0$  as that is just asking for existence of any subcomplex that is homeomorphic to the 2-sphere. We consider the question of deleting *at most*  $k$  triangles.

DELETION-TO-2-DIM-SPHERE

**Input:** A pair  $(\mathcal{K}, k)$  where  $\mathcal{K}$  is a 2-dimensional simplicial complex and  $k$  is a positive integer.

**Question:** Can we delete at most  $k$  triangles in  $\mathcal{K}$  so that the remaining subcomplex is homeomorphic to the 2-dimensional sphere?

There a simple  $\mathcal{O}(3^k \text{poly}(|\mathcal{K}|))$  time algorithm for this problem: While there is an edge incident with at least three triangles, among any three of these triangles at least one must be deleted. Recursive branching on these configurations gives rise to search tree with at most  $3^k$  leaves, each of which is an instance with (1)  $k = 0$  and at least one edge is shared by at least three triangles, or (2)  $k \geq 0$  and each edge is shared by at most two triangles. The former instances can clearly be discarded, the latter can be easily solved in polynomial time: Connected components with a boundary can be discarded (updating budget  $k$  accordingly); connected components without boundary have each edge being shared by exactly two triangles and we can efficiently test which ones are homeomorphic to the 2-sphere (keeping the largest).

Knowing, thus, that DELETION-TO-2-DIM-SPHERE is *fixed-parameter tractable* for parameter  $k$ , we ask whether it admits a polynomial kernelization or compression, i.e., an efficient preprocessing algorithm that returns an equivalent instance of size polynomial in  $k$ . We prove that this is the case by giving, in particular, a compression to almost linear bit-size.

► **Theorem 3.** *The DELETION-TO-2-DIM-SPHERE problem admits a polynomial kernelization to instances with  $\mathcal{O}(k^2)$  triangles and bit-size  $\mathcal{O}(k^2 \log k)$  and a polynomial compression to weighted instances with  $\mathcal{O}(k)$  triangles and bit-size  $\mathcal{O}(k \log k)$ .*

**Related work.** A sketch of NP-hardness for the 2-DIM-SPHERE problem was given by Ivanov [11] in a Mathoverflow question.

Our work is one of the few ones combining topology and fixed parameter tractability. In this direction there have been recent results focused on algorithms in 3-manifold topology [2, 4, 5, 6, 13]. The problem of finding a shortest 1-dimensional cycle  $\mathbb{Z}_2$ -homologous to a given cycle in a 2-dimensional cycle was shown to be NP-hard by Chao and Freedman [8]. Erickson and Nayyeri [9] showed that the problem is fixed-parameter tractable for surfaces, when parameterized by genus of the surface. The result has been extended [7] to arbitrary 2-dimensional simplicial complexes parameterized by the first Betti number. Finally, let us mention that deciding whether a graph (1-dimensional simplicial complex) can be embedded in surface of genus  $g$  is fixed-parameter tractable with respect to the genus [12, 16].

**Organization.** We begin with preliminaries on computational topology and parameterized complexity (Section 2). The proofs for Theorems 1 and 2 about 2-DIM-SPHERE are given in Section 3 and 4. The preprocessing result for DELETION-TO-2-DIM-SPHERE, i.e., Theorem 3, is proved in Section 5. We conclude in Section 6 with some open problems.

## 2 Background and notation

For each positive integer  $n$  we use  $[n]$  to describe the set  $\{1, \dots, n\}$ .

**Topological background.** We give a very succinct summary of the topological background we need and refer the reader to [15, Chapter 1] or [17, Chapter 1] for a comprehensive introduction. The results we mention are standard and available in several books.

A *homeomorphism* between two topological spaces is a continuous mapping between the two spaces whose inverse is also continuous. If such a homeomorphism exists, we say that the two spaces are *homeomorphic*. Any topological property is invariant under homeomorphisms.

A *d-manifold* is a topological space where each point has a neighborhood homeomorphic to  $\mathbb{R}^d$  or the closed half-space  $\{(x_1, \dots, x_d) \in \mathbb{R}^d \mid x_1 \geq 0\}$ . A point of the manifold where no neighborhood is homeomorphic to  $\mathbb{R}^d$  is a *boundary point*. In this paper we focus on 2-manifolds, often called *surfaces*, which are locally equivalent to the Euclidean plane or a half-plane. It is known that the boundary of a (compact) 2-manifold is the union of finitely many 1-manifolds (circles). A surface can be described by a collection of triangles and a collection of pairs of edges of triangles that are identified. If each edge appears in some pairing, then the surface has no boundary.

A *geometric d-simplex* is the convex hull of  $d + 1$  points in  $\mathbb{R}^{d'}$  that are not contained in any hyperplane of dimension  $d - 1$ ; this requires  $d' \geq d$ . A *face* of simplex  $\sigma$  is a simplex of a subset of the points defining  $\sigma$ . A *geometric simplicial complex*  $\mathcal{K}$  is a collection of geometric simplices where each face of each simplex of  $\mathcal{K}$  is also in  $\mathcal{K}$ , and any non-empty intersection of any two simplices of  $\mathcal{K}$  is also in  $\mathcal{K}$ . The *carrier* of  $\mathcal{K}$ , denoted by  $\|\mathcal{K}\|$ , is the union of all the simplices in  $\mathcal{K}$ . A geometric simplicial complex  $\mathcal{K}$  is a *triangulation* of  $X$  if  $X$  and  $\|\mathcal{K}\|$  are homeomorphic. Quite often we talk about properties of  $\mathcal{K}$  when we mean properties of its carrier  $\|\mathcal{K}\|$ . For example, we may say that a geometric simplicial complex  $\mathcal{K}$  is homeomorphic to a topological space  $X$  when we mean that  $\|\mathcal{K}\|$  and  $X$  are homeomorphic.

An (*abstract*) *simplicial complex*  $\mathcal{K}$  is a finite family of sets with the property that any subset of any set of  $\mathcal{K}$  is also contained in  $\mathcal{K}$ . An example of abstract simplicial complex is  $\{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{3, 4\}, \{1, 3, 4\}\}$ . The singletons of  $\mathcal{K}$  are called *vertices* and the set of vertices is denoted by  $V(\mathcal{K})$ . We can assume without loss of generality that  $V(\mathcal{K}) = [n]$  for some natural number  $n$ , as we already had in the previous example. The *dimension* of the (abstract) simplicial complex  $\mathcal{K}$  is  $\max_{\sigma \in \mathcal{K}} |\sigma| - 1$ .

In this paper we focus on (abstract) simplicial complexes and *we will remove the adjective "abstract"* when referring to them. Here we are interested in 2-dimensional simplicial complexes. We can describe them by giving either the list of all simplices or a list of the inclusion-wise maximal simplices. Since in dimension 2 the length of these two lists differ by a constant factor, the choice is asymptotically irrelevant. (For unbounded dimensions, this difference is sometimes relevant.)

A *geometric realization* of a simplicial complex  $\mathcal{K}$  is an injective mapping  $f: V(\mathcal{K}) \rightarrow \mathbb{R}^{d'}$  such that  $\{CH(f(\sigma)) \mid \sigma \in \mathcal{K} \setminus \{\emptyset\}\}$  is a geometric simplicial complex, where  $CH(\cdot)$  denotes the convex hull. It is easy to show that the carriers of any two geometric realizations of a simplicial complex are homeomorphic. Abusing terminology, we will talk about properties of a simplicial complex when (the carrier of) its geometric realizations have the property. For example, we say that a simplicial complex  $\mathcal{K}$  is triangulation of the 2-sphere when we mean that some geometric realization of  $\mathcal{K}$  is a triangulation of the 2-sphere (and thus all geometric realizations of  $\mathcal{K}$  are triangulations of the 2-sphere).

**Parameterized complexity.** A *parameterized problem* is a language  $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$  where  $\Sigma$  is any finite alphabet and  $\mathbb{N}$  denotes the non-negative integers; the second component  $k$  of an instance  $(x, k) \in \Sigma^* \times \mathbb{N}$  is called its *parameter*. A parameterized problem  $\mathcal{Q}$  is *fixed-parameter tractable* if there is an algorithm  $A$ , a constant  $c$ , and a computable function  $f: \mathbb{N} \rightarrow \mathbb{N}$  such that  $A$  correctly decides  $(x, k) \in \mathcal{Q}$  in time  $f(k) \cdot |x|^c$  for all  $(x, k) \in \Sigma^* \times \mathbb{N}$ . A *kernelization* of a parameterized problem  $\mathcal{Q}$  with *size*  $h: \mathbb{N} \rightarrow \mathbb{N}$  is a polynomial-time algorithm  $K$  that on input  $(x, k) \in \Sigma^* \times \mathbb{N}$  takes time polynomial in  $|x| + k$  and returns an instance  $(x', k')$  of size at most  $h(k)$  such that  $(x, k) \in \mathcal{Q}$  if and only if  $(x', k') \in \mathcal{Q}$ . If  $h(k)$  is polynomially bounded then  $K$  is a *polynomial kernelization*. If the output of  $K$  is instead an instance of any (unparameterized) problem  $L'$  then we called it a (*polynomial*) *compression*.

The prevalent method of showing that a parameterized problem  $\mathcal{Q}' \subseteq \Sigma'^* \times \mathbb{N}$  is *not* fixed-parameter tractable is to give a *parameterized reduction* from a problem  $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$  that is hard for a class called  $W[1]$ , which contains the class FPT of all fixed-parameter tractable problems; it is assumed that  $FPT \neq W[1]$ . A parameterized reduction from  $\mathcal{Q}$  to  $\mathcal{Q}'$  is an algorithm  $R$  that on input  $(x, k) \in \Sigma^* \times \mathbb{N}$  takes time  $f(k) \cdot |x|^c$  and returns an instance  $(x', k') \in \Sigma'^* \times \mathbb{N}$  such that:  $(x, k) \in \mathcal{Q}$  if and only if  $(x', k') \in \mathcal{Q}'$  and such that  $k' \leq g(k)$ ; here  $f, g: \mathbb{N} \rightarrow \mathbb{N}$  are computable functions and  $c$  is a constant, all independent of  $(x, k)$ . Parameterized reductions can also be used to transfer lower bounds on the running time. A common starting point for this is the *Exponential Time Hypothesis (ETH)* which posits that there is a constant  $\delta_3 > 0$  such that no algorithm solves 3-SAT in time  $\mathcal{O}(2^{\delta_3 n})$  where  $n$  denotes the number of variables. In particular, this rules out subexponential-time algorithms for 3-SAT and, by appropriate reductions, for a host of other problems.

### 3 Hardness of 2-dim sphere

In this section we provide a proof for Theorem 1, namely that 2-DIM-SPHERE is  $W[1]$ -hard for parameter  $k$  and, under ETH, admits no  $\mathcal{O}(n^{o(\sqrt{k})})$  time algorithm. To obtain the result we give a polynomial-time reduction from the GRID TILING problem introduced by Marx [14].

GRID TILING

**Input:** A triple  $(n, k, \mathcal{S})$  where  $n$  is a positive integer,  $k$  is a positive integer, and  $\mathcal{S}$  is a tuple of  $k^2$  nonempty sets  $S_{i,j} \subseteq [n] \times [n]$ , where  $i, j \in [k]$ .

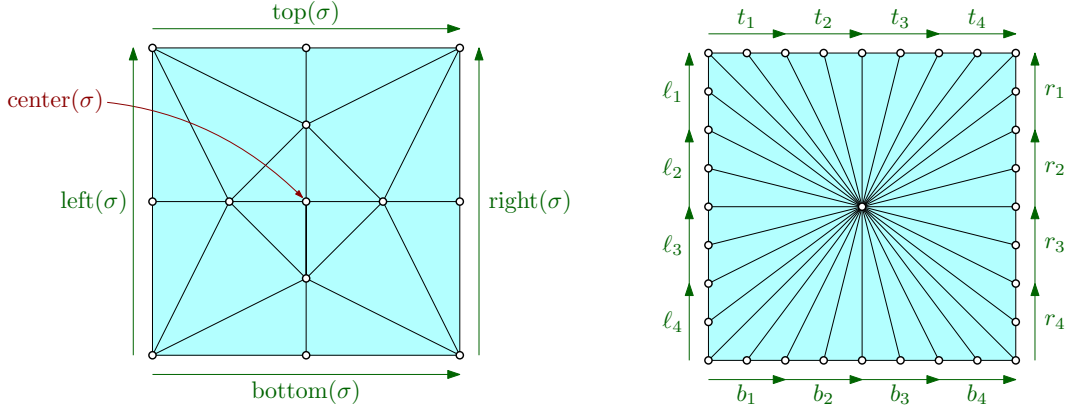
**Question:** Can we choose for each  $i, j \in [k]$  a pair  $(a_{i,j}, b_{i,j}) \in S_{i,j}$  such that  $a_{i,j} = a_{i,j+1}$  for all  $i \in [k], j \in [k-1]$ , and  $b_{i,j} = b_{i+1,j}$  for all  $i \in [k-1], j \in [k]$ ?

It is convenient to visualize the input elements as displayed in a  $(k \times k)$ -tiled square. The squares are indexed like matrices: the top left tile corresponds to the index  $(i, j) = (1, 1)$  and the bottom left tile corresponds to the index  $(i, j) = (k, 1)$ . Inside the  $(i, j)$ -tile we put the elements of  $S_{i,j}$ . The task is to select a 2-tuple in each tile such that the selected elements in each row have the same first coordinate and the selected elements in each column have the same second coordinate. The following lower bound is known for GRID TILING.

► **Theorem 4** ([14]). GRID TILING is  $W[1]$ -hard and, unless ETH fails, it has no  $f(k)n^{o(k)}$ -time algorithm for any computable function  $f$ .

Consider an instance  $(n, k, \mathcal{S})$  of GRID TILING. We are going to construct an equivalent instance  $(\mathcal{K}, k')$  to 2-DIM-SPHERE where  $k' = \Theta(k^2)$ .

Let  $\sigma$  be the simplicial complex shown in Figure 1, left. It is a triangulation of a square with a middle vertex, denoted  $\text{center}(\sigma)$ . We denote the consecutive 2-edge paths on the boundary as  $\text{left}(\sigma)$ ,  $\text{top}(\sigma)$ ,  $\text{right}(\sigma)$  and  $\text{bottom}(\sigma)$ . The orientation of the path, indicated



■ **Figure 1** Left: The triangulated square  $\sigma$ . Right: back sheet when  $k = 4$ .

with an arrow, defines the way we glue in later steps of the construction. In our figures we will always orient the squares to match these names in the intuitive way. For our construction, the important property of  $\sigma$  is that there is no triangle containing  $\text{center}(\sigma)$  and one boundary edge and that there is no triangle containing two boundary edges of  $\sigma$ .

For each  $(a, b)$  in each  $S_{i,j}$  we make a new copy of  $\sigma$  and denote it by  $\sigma(a, b, i, j)$ . We make some identifications, according to the following rules:

- For each  $i \in [k]$ ,  $j \in [k-1]$ , and  $a \in [n]$ , we identify together all the 2-edge paths  $\text{right}(\sigma(a, b, i, j))$ , where  $(a, b) \in S_{i,j}$ , and all the 2-edge paths  $\text{left}(\sigma(a', b', i, j+1))$ , where  $(a', b') \in S_{i,j+1}$ . Thus, for each  $i, j, a$  we have identified  $|\{b \in [n] \mid (a, b) \in S_{i,j}\}| + |\{b' \in [n] \mid (a', b') \in S_{i,j+1}\}|$  2-edge paths into a single one.
- For each  $i \in [k-1]$ ,  $j \in [k]$ , and  $b \in [n]$ , we identify together all the 2-edge paths  $\text{bottom}(\sigma(a, b, i, j))$ , where  $(a, b) \in S_{i,j}$ , and all the 2-edge paths  $\text{top}(\sigma(a', b, i+1, j))$ , where  $(a', b) \in S_{i+1,j}$ . Thus, for each  $i, j, b$  we have identified  $|\{a \in [n] \mid (a, b) \in S_{i,j}\}| + |\{a' \in [n] \mid (a', b) \in S_{i+1,j}\}|$  2-edge paths into a single one.
- For each  $i, j \in [n]$ , we identify the vertices  $\text{center}(\sigma(a, b, i, j))$  over all  $(a, b) \in S_{i,j}$ . Thus, we identified  $|S_{i,j}|$  vertices into a single one.

To finalize the construction, we triangulate a square such that it has  $2k$  edges on each side, as shown in Figure 1, right. We will refer to this simplicial complex as the *back sheet*. We split the boundary of the square into 2-edge paths and label them, in a clockwise traversal of the boundary of the square, by  $t_1, \dots, t_k, r_1, \dots, r_k, b_k, \dots, b_1, \ell_k, \dots, \ell_1$ . (We use  $t$  as intuition for top,  $r$  as intuition for right, etc.) Note that the indices for  $b$  and  $\ell$  run backwards. In the figure we also indicate the orientation of the 2-edge paths, that are relevant for the forthcoming identifications.

Then we make the following additional identifications.

- For each  $i \in [n]$  and each  $(a, b) \in S_{i,1}$ , we identify  $\text{left}(\sigma(a, b, i, 1))$  and  $\ell_i$ .
- For each  $i \in [n]$  and each  $(a, b) \in S_{i,k}$ , we identify  $\text{right}(\sigma(a, b, i, k))$  and  $r_i$ .
- For each  $j \in [n]$  and each  $(a, b) \in S_{1,j}$ , we identify  $\text{top}(\sigma(a, b, 1, j))$  and  $t_j$ .
- For each  $j \in [n]$  and each  $(a, b) \in S_{k,j}$ , we identify  $\text{bottom}(\sigma(a, b, k, j))$  and  $b_j$ .

Note that whenever we identify the endpoints of two edges in pairs, we also identified the edges. Thus, we have constructed a simplicial complex. (In any case, we could always use barycentric subdivisions to ensure that we have a simplicial complex.) Let  $\mathcal{K} = \mathcal{K}(n, k, \mathcal{S})$  denote the resulting simplicial complex. Set  $k' = 16 \cdot k^2 + 8k$ . With the following lemmas we prove that  $(\mathcal{K}, k')$  is yes for 2-DIM-SPHERE if and only if  $(n, k, \mathcal{S})$  is yes for GRID TILING.

► **Lemma 5** ( $*^1$ ). *If  $(n, k, \mathcal{S})$  is a yes-instance for GRID TILING, then  $\mathcal{K}$  contains a subcomplex with  $k'$  triangles that is homeomorphic to the 2-sphere.*

**Proof.** Since  $(n, k, \mathcal{S})$  is a yes-instance for GRID TILING, there are pairs  $(a_{i,j}, b_{i,j}) \in S_{i,j}$ , where  $i, j \in [k]$ , such that  $a_{i,j} = a_{i,j+1}$  (for  $i \in [k], j \in [k-1]$ ) and  $b_{i,j} = b_{i+1,j}$  (for  $i \in [k-1], j \in [k]$ ).

Consider the subcomplex  $\tilde{\mathcal{K}}$  of  $\mathcal{K}$  induced by the squares  $\sigma(a_{i,j}, b_{i,j}, i, j)$ , where  $i, j \in [k]$ . During the identifications we have glued  $\sigma(a_{i,j}, b_{i,j}, i, j)$  to  $\sigma(a_{i,j+1}, b_{i,j+1}, i, j+1)$  when making the identification  $\text{right}(\sigma(a_{i,j}, b_{i,j}, i, j)) = \text{left}(\sigma(a_{i,j+1}, b_{i,j+1}, i, j+1))$  because  $a_{i,j} = a_{i,j+1}$  (for  $i \in [k], j \in [k-1]$ ). Similarly, we have glued  $\sigma(a_{i,j}, b_{i,j}, i, j)$  to  $\sigma(a_{i+1,j}, b_{i+1,j}, i+1, j)$  when making the identification  $\text{bottom}(\sigma(a_{i,j}, b_{i,j}, i, j)) = \text{top}(\sigma(a_{i+1,j}, b_{i+1,j}, i+1, j))$  because  $b_{i,j} = b_{i+1,j}$  (for  $i \in [k-1], j \in [k]$ ). Thus  $\tilde{\mathcal{K}}$  is a "big square" obtained by glueing  $k^2$  copies of  $\sigma$  in a  $(k \times k)$ -grid-like way. Together with the back sheet, that is glued to the boundary of  $\tilde{\mathcal{K}}$ , we get a triangulation of the 2-sphere. Since each square  $\sigma(\cdot)$  has 16 triangles and the back sheet has  $8k$  triangles, the resulting triangulation has  $16k^2 + 8k$  triangles. A formal argument to prove this can also be carried out using Euler's formula. ◀

► **Lemma 6.** *If  $\mathcal{K}$  contains a subcomplex  $\mathcal{K}'$  homeomorphic to the 2-sphere, then  $(n, k, \mathcal{S})$  is a yes-instance for GRID TILING.*

**Proof.** We show this by first demonstrating that, for any pair  $i, j \in [k]$ , the subcomplex  $\mathcal{K}'$  cannot contain two distinct squares  $\sigma(a, b, i, j)$  and  $\sigma(c, d, i, j)$ . We then show that for any pair  $i, j \in [k]$ , at least one of the squares  $\sigma(a, b, i, j)$  must be part of  $\mathcal{K}'$ . Lastly we combine these two facts to construct a solution for the GRID TILING instance  $(n, k, \mathcal{S})$ .

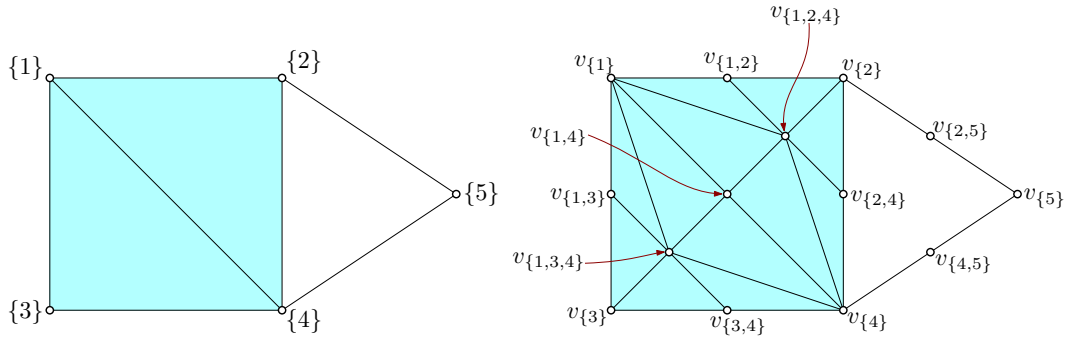
We begin by noting that  $\mathcal{K}'$  cannot be empty, and that if for any values of  $a, b, i, j$ , the subcomplex  $\mathcal{K}'$  contains one triangle from  $\sigma(a, b, i, j)$ , then  $\mathcal{K}'$  must contain all triangles from  $\sigma(a, b, i, j)$ . This follows from the fact that the 2-dimensional sphere has no boundary. In the rest of this argument, we need only consider whether  $\mathcal{K}'$  does or does not contain all of  $\sigma(a, b, i, j)$  for any values of  $a, b, i, j$ .

Now assume that, for some pair  $i, j \in [k]$ , the subcomplex  $\mathcal{K}'$  contains two distinct squares  $\sigma(a, b, i, j)$  and  $\sigma(c, d, i, j)$  and consider the neighborhood of  $\text{center}(\sigma(a, b, i, j))$ . Recalling that  $\text{center}(\sigma(a, b, i, j))$  is identified with  $\text{center}(\sigma(c, d, i, j))$ , we see that this point has no neighborhood homeomorphic to a plane, and so clearly  $\mathcal{K}'$  cannot contain both of these distinct squares.

We now show that, for any pair  $i, j \in [k]$ ,  $\mathcal{K}'$  must contain some square of the form  $\sigma(a, b, i, j)$ . If  $\mathcal{K}'$  contains no squares  $\sigma(\cdot)$  at all then it can only contain either part of, or the whole of, the back sheet but either way  $\mathcal{K}'$  cannot be a 2-sphere. Thus we know that  $\mathcal{K}'$  must contain  $\sigma(a, b, i, j)$  for at least one set of values  $a, b, i, j$ . Given this, assume we have a pair  $i', j' \in [n]$  such that  $\sigma(a', b', i', j')$  is not in  $\mathcal{K}'$  for each pair  $(a', b') \in S_{i', j'}$ . Let  $\sigma(a, b, i, j)$  be a square in  $\mathcal{K}'$  for some pair  $a, b$ , and without loss of generality, assume that  $i' = i - 1$  and  $j' = j$ . This is equivalent to choosing two adjacent cells where one contains a square in  $\mathcal{K}'$  and the other does not contain any square in  $\mathcal{K}'$ , and can always be achieved by appropriate selection of values (and possibly rotating or flipping the whole construction).

Consider an edge of the 2-edge path  $\text{top}(\sigma(a, b, i, j))$  in  $\mathcal{K}'$ . Since  $\mathcal{K}'$  is a 2-dimensional sphere, this edge cannot be a boundary and thus must separate two distinct triangles. One of these triangles is present in  $\sigma(a, b, i, j)$ . By our construction, the other triangle is either in  $\sigma(a', b, i - 1, j)$  (where  $\text{top}(\sigma(a, b, i, j))$  is identified with  $\text{bottom}(\sigma(a', b, i - 1, j))$ ) or in

<sup>1</sup> (Full) proofs of statements marked \* will appear in the journal version of this paper.



■ **Figure 2** A simplicial complex  $\mathcal{K}$  (left side) and its barycentric subdivision  $\text{Sd}(\mathcal{K})$  (right side).

$\sigma(c, b, i, j)$  with  $a \neq c$  (where  $\text{top}(\sigma(a, b, i, j))$  is identified with  $\text{top}(\sigma(c, b, i, j))$ ). This means that one of  $\sigma(c, b, i, j)$  or  $\sigma(a', b, i - 1, j)$  must be in  $\mathcal{K}'$ . By our earlier argument,  $\sigma(a, b, i, j)$  and  $\sigma(c, b, i, j)$  cannot both be in  $\mathcal{K}'$ . This means that  $\sigma(a', b, i - 1, j)$  must be in  $\mathcal{K}'$ , and thus our assumption must be false. Therefore for each pair  $i, j \in [n]$ , at least one square  $\sigma(a, b, i, j)$  must be in the subcomplex  $\mathcal{K}'$ .

Combining these results we see that if the subcomplex  $\mathcal{K}'$  is a 2-sphere, then  $\mathcal{K}'$  contains exactly one square  $\sigma(a_{i,j}, b_{i,j}, i, j)$  for each pair  $i, j \in [n]$ . Since for each values  $i, j, a, b$  we have that  $\sigma(a_{i,j}, b_{i,j}, i, j) \in \mathcal{K}$  if and only if  $(a_{i,j}, b_{i,j}) \in S_{i,j}$ , we obtain that  $(a_{i,j}, b_{i,j}) \in S_{i,j}$  for each  $i, j \in [k]$ . As  $\text{top}(\sigma(a, b, i, j))$  is identified with  $\text{bottom}(\sigma(a', b, i - 1, j))$ , by induction we see that, for each  $j \in [k]$ , we have  $b_{1,j} = b_{2,j} = \dots = b_{k,j}$ . A similar argument shows that for each  $i \in [k]$  we have  $a_{i,1} = a_{i,2} = \dots = a_{i,k}$ . We deduce that  $(a_{i,j}, b_{i,j})$ , for each pair  $i, j \in [k]$ , is a solution for  $(n, k, \mathcal{S})$ . ◀

Lemmas 5 and 6 establish correctness of our reduction from GRID TILING to 2-DIM-SPHERE. Clearly, the reduction can be performed in polynomial time, and the parameter value of a created instance is  $\mathcal{O}(k^2)$ . Thus, Theorem 1 now follows directly from Theorem 4.

#### 4 A tight algorithm for 2-dim-sphere

For each simplicial complex  $\mathcal{K}$ , let  $\text{Sd}(\mathcal{K})$  be its barycentric subdivision. Its construction for the 2-dimensional case is as follows (see also Figure 2). Each vertex, edge and triangle of  $\mathcal{K}$  is a vertex of  $\text{Sd}(\mathcal{K})$ . To emphasize the difference, for a simplex  $\tau$  of  $\mathcal{K}$  we use  $v_\tau$  for the corresponding vertex in  $\text{Sd}(\mathcal{K})$ . There is an edge  $v_\tau v_{\tau'}$  in  $\text{Sd}(\mathcal{K})$  between any two simplices  $\tau$  and  $\tau'$  of  $\mathcal{K}$  precisely when one is contained in the other. There is a triangle  $v_{\tau_1} v_{\tau_2} v_{\tau_3}$  in  $\text{Sd}(\mathcal{K})$  whenever there is a chain of inclusions  $\tau_1 \subsetneq \tau_2 \subsetneq \tau_3$ . It is well-known, and not difficult to see, that  $\text{Sd}(\mathcal{K})$  and  $\mathcal{K}$  are homeomorphic. See for example [15, Chapter 1] or [17, Chapter 2]. Let  $\text{Sd}_1(\mathcal{K})$  be the 1-skeleton of  $\text{Sd}(\mathcal{K})$ , which is a graph.

An *isomorphism* between two simplicial complexes  $\mathcal{K}_1$  and  $\mathcal{K}_2$  is a bijective map  $f: V(\mathcal{K}_1) \rightarrow V(\mathcal{K}_2)$  such that, for all  $\{v_1, \dots, v_k\} \subseteq V(\mathcal{K}_1)$ , the simplex  $\{v_1, v_2, \dots, v_k\}$  is in  $\mathcal{K}_1$  precisely when  $\{f(v_1), f(v_2), \dots, f(v_k)\}$  is a simplex of  $\mathcal{K}_2$ . Two simplicial complexes are *isomorphic* if and only if there exists some isomorphism between them. When two simplicial complexes are isomorphic, then they are also homeomorphic. (We can make geometric realizations for both simplicial complexes with the same carrier.) Note that isomorphism of simplicial complexes of dimension 1 matches the definition of isomorphism of graphs.

Testing isomorphism of simplicial complexes can be reduced to testing isomorphism of *colored* graphs, as follows. Let  $G$  and  $H$  be graphs and assume that we have colorings



$c_G: V(G) \rightarrow \mathbb{N}$  and  $c_H: V(H) \rightarrow \mathbb{N}$ . (The term coloring here refers to a labeling; there is no relation to the standard graph colorings.) A *color-preserving isomorphism* between  $(G, c_G)$  and  $(H, c_H)$  is a graph isomorphism  $f: V(G) \rightarrow V(H)$  such that, for each vertex  $v \in V(G)$ , it holds  $c_H(f(v)) = c_G(v)$ . Thus, the isomorphism preserves the color of each vertex. We say that  $(G, c_G)$  and  $(H, c_H)$  are *color-preserving isomorphic* if there is some color-preserving isomorphism between them. We will use the dimension  $\dim$  of the simplex as the coloring for the graph  $\text{Sd}_1(\cdot)$ . Thus  $\dim(v_\tau) = |\tau| - 1$  for each simplex  $\tau$  of the simplicial complex.

► **Lemma 7** (\*). *Two simplicial complexes  $\mathcal{K}_1$  and  $\mathcal{K}_2$  are isomorphic if and only if  $(\text{Sd}_1(\mathcal{K}_1), \dim)$  and  $(\text{Sd}_1(\mathcal{K}_2), \dim)$  are color-preserving isomorphic.*

► **Lemma 8**. *Let  $\mathcal{K}_1$  and  $\mathcal{K}_2$  be simplicial complexes. The simplicial complex  $\mathcal{K}_1$  has a subcomplex isomorphic to  $\mathcal{K}_2$  if and only if  $\text{Sd}_1(\mathcal{K}_1)$  has a subgraph  $G$  such that  $(G, \dim)$  and  $(\text{Sd}_1(\mathcal{K}_2), \dim)$  are color-preserving isomorphic.*

**Proof.** Note that for each subcomplex  $\mathcal{K}'_1$  of  $\mathcal{K}_1$  we have that  $\text{Sd}_1(\mathcal{K}'_1)$  is exactly the subgraph of  $\text{Sd}_1(\mathcal{K}_1)$  induced by the vertices  $v_\tau$ , for  $\tau \in \mathcal{K}'_1 \setminus \{\emptyset\}$ . Therefore, if  $\mathcal{K}_1$  has a subcomplex  $\mathcal{K}'_1$  isomorphic to  $\mathcal{K}_2$ , then the graph  $G = \text{Sd}_1(\mathcal{K}'_1)$  is a subgraph of  $\text{Sd}_1(\mathcal{K}_1)$  and, by Lemma 7,  $(G, \dim)$  and  $(\text{Sd}_1(\mathcal{K}_2), \dim)$  are color-preserving isomorphic.

Assume, for the other direction, that  $(G, \dim)$  and  $(\text{Sd}_1(\mathcal{K}_2), \dim)$  are color-preserving isomorphic for some subgraph  $G$  of  $\text{Sd}_1(\mathcal{K}_1)$ . Let  $f$  be such a color-preserving isomorphism. First we show that  $G$  is  $\text{Sd}_1(\mathcal{K}'_1)$  for some subcomplex  $\mathcal{K}'_1$  of  $\mathcal{K}_1$ . Indeed, consider any vertex  $v_\tau$  of  $G$  such that for no superset  $\tilde{\tau}$  of  $\tau$  we have  $v_{\tilde{\tau}}$  in  $G$ . The vertex  $f(v_\tau)$  is a vertex  $v_{\tau_2}$  for  $\tau_2 \in \mathcal{K}_2$  and moreover  $|\tau| = |\tau_2|$  as  $f$  preserves color and therefore dimension. Each subset  $\tau'_2$  of  $\tau_2$  has some vertex  $v_{\tau'_2}$  in  $\text{Sd}_1(\mathcal{K}_2)$ . For each such  $\tau'_2 \subset \tau_2$  we have some distinct vertex  $v_{\tau'_1}$  in  $G$  such that  $f(v_{\tau'_1}) = v_{\tau'_2}$  and  $v_{\tau'_1}$  must be adjacent to  $v_\tau$ . Since  $\tau$  and  $\tau_2$  have the same cardinality, they have the same number of subsets, and thus  $\tau'_1$  iterates over all subsets of  $\tau$ , when  $\tau'_2$  iterates over the subsets of  $\tau_2$ . This means that  $v_{\tau'}$  is in  $G$  for all subsets  $\tau' \subset \tau$ . Therefore, if we take  $\mathcal{K}'_1 = \{\tau \in \mathcal{K}_1 \mid v_\tau \in V(G)\} \cup \{\emptyset\}$ , then  $\mathcal{K}'_1$  is a simplicial complex and  $G = \text{Sd}_1(\mathcal{K}'_1)$ . From Lemma 7 it follows that  $\mathcal{K}'_1$  and  $\text{Sd}_1(\mathcal{K}_2)$  are isomorphic. ◀

► **Lemma 9**. *Let  $\mathcal{K}$  be a simplicial complex with  $n$  simplices and let  $\mathcal{K}'$  be a simplicial complex with  $k$  simplices. Let  $t$  be the treewidth of  $\text{Sd}_1(\mathcal{K}')$ . In time  $n^{\mathcal{O}(t)}$  we can decide whether  $\mathcal{K}$  contains a subcomplex isomorphic to  $\mathcal{K}'$ .*

**Proof.** Alon, Yuster and Zwick [1, Theorem 6.3] show how to find in a graph  $G$  a subgraph isomorphic to a given graph  $H$  in time  $2^{\mathcal{O}(|V(H)|)} |V(G)|^{\mathcal{O}(t_H)}$ , where  $t_H$  is the treewidth of  $H$ . The technique is color-coding. For this, one tries several different colorings of the vertices of  $G$  with  $|V(H)|$  colorings, and then uses dynamic programming to search for a copy of  $H$  in  $G$  where all the colors of the vertices are distinct. Thus, if the vertices of  $H$  and  $G$  are already classified into some classes, then this can only help the algorithm. The class of a vertex can be considered as part of the coloring. This means that the algorithm can be trivially adapted to the problem of subgraph color-preserving isomorphism: given two pairs  $(G, c_G)$  and  $(H, c_H)$ , where  $c_G$  and  $c_H$  are colorings of the vertices, is there a subgraph  $G'$  of  $G$  such that  $(G', c_{G'})$  and  $(H, c_H)$  are color-preserving isomorphic, where  $c_{G'}$  is the restriction of  $c_G$  to  $G'$ .

Because of Lemma 8, deciding whether  $\mathcal{K}$  contains a subcomplex isomorphic to  $\mathcal{K}'$  is equivalent to deciding whether  $\text{Sd}_1(\mathcal{K})$  contains a subgraph  $G$  such that  $(G, \dim)$  and  $(\text{Sd}_1(\mathcal{K}'), \dim)$  are color-preserving isomorphic. Apply the color-coding algorithm of Alon et al. as, discussed before, we spend  $2^{\mathcal{O}(|V(\text{Sd}_1(\mathcal{K}'))|)} |V(\text{Sd}_1(\mathcal{K}))|^{\mathcal{O}(t)} = 2^{\mathcal{O}(k)} n^{\mathcal{O}(t)}$  time. ◀

**Proof of Theorem 2.** There are  $2^{\mathcal{O}(k)}$  different (unlabeled) triangulations of the 2-sphere with at most  $k$  triangles; see for example [20, 3], using that  $k$  triangles entail having at most  $\mathcal{O}(k)$  vertices. For each such triangulation, let  $\mathcal{K}_i$  be the corresponding simplicial complex. Then  $\text{Sd}_1(\mathcal{K}_i)$  is a planar graph with  $\mathcal{O}(k)$  vertices and thus has treewidth  $\mathcal{O}(\sqrt{k})$ . Using Lemma 9 we can decide in  $2^{\mathcal{O}(k)} n^{\mathcal{O}(\sqrt{k})}$  time whether  $\mathcal{K}$  has a subcomplex isomorphic to  $\mathcal{K}_i$ . Iterating over all the  $2^{\mathcal{O}(k)}$  triangulations we spend in total  $2^{\mathcal{O}(k)} \cdot 2^{\mathcal{O}(k)} n^{\mathcal{O}(\sqrt{k})}$  time and the result follows. ◀

## 5 Kernelization and compression for Deletion-to-2-dim-sphere

In this section, we prove that DELETION-TO-2-DIM-SPHERE admits a polynomial kernelization that returns instances with  $\mathcal{O}(k^2)$  triangles and has bit-size  $\mathcal{O}(k^2 \log k)$ , respectively a polynomial compression to a weighted version with bit-size  $\mathcal{O}(k \log k)$ . We first give a few simple reduction rules and then show how to reduce (and possibly encode) the resulting instances. The rules are to be applied in order, i.e., preference is given to earlier rules. Recall that input instances  $(\mathcal{K}, k)$  consist of a 2-dimensional simplicial complex  $\mathcal{K}$  and an integer  $k$ , and ask whether deletion of at most  $k$  triangles from  $\mathcal{K}$  yields a subcomplex that is homeomorphic to the 2-dimensional sphere  $\mathbb{S}_2$ .

In what follows, we will delete subcomplexes from an instance of our problem and at the same time reduce the value of  $k$ . If at any point in time  $k$  becomes negative we know that our original instance was a no-instance, so we will assume that  $k$  is always non-negative. Additionally we point out that whenever deleting a subcomplex from our simplicial complex, any vertices or edges which would no longer be contained in any triangle are also deleted.

► **Reduction Rule 1.** *If any triangle  $T \in \mathcal{K}$  has an edge face that is not a face of any other triangle in  $\mathcal{K}$  then delete  $T$  from  $\mathcal{K}$  and reduce  $k$  by one.*

Clearly, such a triangle  $T$  cannot be contained in a subcomplex  $\mathcal{K}' \subseteq \mathcal{K}$  that is homeomorphic to the 2-sphere, and hence it must be among the  $k$  deleted triangles in any solution (if one exists). Note that when Rule 1 does not apply, each edge in  $\mathcal{K}$  is shared by at least two triangles of  $\mathcal{K}$ . On the other hand, in the desired subcomplex that is homeomorphic with the 2-sphere each edge is shared by *exactly* two triangles. Denote by  $\mathcal{T} \subseteq \mathcal{K}$  the set of triangles that share at least one of their edges with more than one other triangle. There is a simple upper bound for the size of  $\mathcal{T}$  if  $(\mathcal{K}, k)$  is a yes-instance.

► **Proposition 10.** *If  $(\mathcal{K}, k)$  is a yes-instance of DELETION-TO-2-DIM-SPHERE then  $|\mathcal{T}| \leq 7k$ .*

**Proof.** Let  $\mathcal{D}$  be a given solution with at most  $k$  triangles. Each triangle in  $\mathcal{T} \setminus \mathcal{D}$  must share at least one edge with a triangle in  $\mathcal{D}$ . Additionally, each triangle in  $\mathcal{D}$  can share an edge with at most six triangles in  $\mathcal{T} \setminus \mathcal{D}$ , as each of the three edges of a triangle in  $\mathcal{D}$  is shared between at most two triangles of  $\mathcal{T} \setminus \mathcal{D}$ . Thus,  $|\mathcal{T} \setminus \mathcal{D}| \leq 6 \cdot |\mathcal{D}|$ , giving  $|\mathcal{T}| \leq 7k$ . ◀

► **Reduction Rule 2.** *Reject the instance if  $|\mathcal{T}| > 7k$ .*

Observe now that in  $\mathcal{K}_{\mathcal{T}} := \mathcal{K} \setminus \mathcal{T}$  all edges are shared by at most two triangles, and that edges shared with triangles in  $\mathcal{T}$  are only part of *one* triangle in  $\mathcal{K}_{\mathcal{T}}$ . Accordingly, triangles in  $\mathcal{K}_{\mathcal{T}}$  form connected components that can be homeomorphic to, e.g., the 2-sphere or to a punctured disk. Say that the *boundary* of a component is the set of edges  $L$  that are contained in exactly one triangle of the component; these are exactly the edges that participate also in triangles of  $\mathcal{T}$ . We distinguish components according to whether or not they have a boundary.

For any component without boundary the procedure is simple: It cannot have any edge  $L$  in common with a triangle of  $\mathcal{T}$  since then three or more triangles of  $\mathcal{K}$  would share  $L$  and all incident triangles would be in  $\mathcal{T}$ . Accordingly, such a component can only be part of the desired 2-sphere if it itself is homeomorphic to the 2-sphere since it is not connected with other triangles in  $\mathcal{K}$ . The only other option is to delete the entire component since deleting it partially would always leave a boundary.

► **Reduction Rule 3.** *Let  $\mathcal{C}$  be a connected component of  $\mathcal{K}_{\mathcal{T}}$  that has no boundary. If  $\mathcal{C}$  is homeomorphic to the 2-sphere  $\mathbb{S}_2$  and  $|\mathcal{K} \setminus \mathcal{C}| \leq k$  then answer yes (and return  $\mathcal{K} \setminus \mathcal{C}$  as a solution). Else, if  $\mathcal{C}$  is not homeomorphic to the 2-sphere or if  $|\mathcal{K} \setminus \mathcal{C}| > k$ , then delete all triangles of  $\mathcal{C}$  from  $\mathcal{K}$  and reduce  $k$  by  $|\mathcal{C}|$ .*

Using Rules 1 through 3 we either solve the instance or we arrive at the situation where  $|\mathcal{T}| \leq 7k$  and all components of  $\mathcal{K}_{\mathcal{T}} = \mathcal{K} \setminus \mathcal{T}$  have boundaries. Observe that, among these, we can safely delete each component  $\mathcal{C}$  that is not homeomorphic to a (punctured) disk: While  $\mathcal{C}$  may contain subcomplexes that are homeomorphic to a (punctured) disk, such a subcomplex cannot be extended to a subcomplex of  $\mathcal{K}$  that is homeomorphic to the 2-sphere since the requirement of having two triangles incident with each edge implies using all triangles of  $\mathcal{C}$ .

► **Reduction Rule 4.** *If  $\mathcal{C}$  is a connected component of  $\mathcal{K}_{\mathcal{T}}$  that has a boundary but is not homeomorphic to a (punctured) disk then delete all triangles of  $\mathcal{C}$  from  $\mathcal{K}$  and reduce  $k$  by  $|\mathcal{C}|$ .*

It remains to consider the case where  $|\mathcal{T}| \leq 7k$  and all components of  $\mathcal{K}_{\mathcal{T}}$  (have boundaries and) are homeomorphic to (punctured) disks. As a first step, let us observe an upper bound on the total length of all component boundaries (in terms of number of edges) for yes-instances.

► **Proposition 11.** *If  $(\mathcal{K}, k)$  is a yes-instance of DELETION-TO-2-DIM-SPHERE then the total length of all boundaries of components of  $\mathcal{K}_{\mathcal{T}}$  is at most  $21k$ .*

**Proof.** By Rule 1 each boundary edge of a component of  $\mathcal{K}_{\mathcal{T}} = \mathcal{K} \setminus \mathcal{T}$  is incident with at least two triangles of  $\mathcal{K}$ , and hence with at least one triangle of  $\mathcal{T}$ . The upper bound of  $3 \cdot |\mathcal{T}| \leq 21k$  follows. ◀

Note that from the upper bound of  $21k$  for the total boundary length we immediately get an upper bound of  $7k$  for the number of components of  $\mathcal{K}_{\mathcal{T}}$  since each component with a boundary must have at least three boundary edges. To get an upper bound on the number of triangles it now suffices to replace large components by “equivalent” ones without changing the status of the instance using Proposition 12. This has two vital aspects: (1) Replaced components must have same boundary and topology. (2) We must avoid creating false positives, as smaller components can be deleted at a lower cost. In Proposition 12 we show how components with boundary length  $\ell$  can be replaced by equivalent ones with  $\mathcal{O}(\ell)$  triangles, addressing (1), and later give two options for addressing (2).

► **Proposition 12 (\*).** *Given a simplicial complex  $\mathcal{K}$  of a punctured sphere where  $\mathcal{K}$  contains  $\ell$  boundary edges, there exists a simplicial complex  $\mathcal{K}'$  such that the following hold:*

1.  $\mathcal{K}'$  contains  $\mathcal{O}(\ell)$  triangles,
2.  $\mathcal{K}$  is homeomorphic to  $\mathcal{K}'$ ,
3. there exists an isomorphism  $f: \partial(\mathcal{K}) \mapsto \partial(\mathcal{K}')$ , and
4. if  $a, b$  are edges of  $\mathcal{K}$  such that  $a, b \in \partial(\mathcal{K})$  and there exists a triangle  $t$  of  $\mathcal{K}$  such that  $a, b \in t$ , then there exists a triangle  $t' \in \mathcal{K}'$  such that  $f(a), f(b) \in t'$ .

To avoid false positives we have two options. First, we can store for each component its initial number of triangles, i.e., the cost for deleting it entirely, noting that costs larger than  $k$  can be replaced by  $k + 1$ . (Recall that partially deleting a component is infeasible.) The output would then be an instance of a weighted version of the problem, and we could encode it using  $\mathcal{O}(k \log k)$  bits, where the log-factor is needed to encode costs in binary and to represent a list of the triangles including vertex names. (We could also assign a larger cost to one triangle per component such that the total is equal to the original value.)

Second, we could apply the replacement only to components with more than  $k$  triangles, and afterwards increase their size to  $k + \mathcal{O}(1)$  by adding additional triangles. Since budget of  $k$  does not allow the deletion of large components, this yields an equivalent instance. The total number of triangles per component is then  $\mathcal{O}(k)$ , and  $\mathcal{O}(k^2)$  for the entire instance; this can be encoded in  $\mathcal{O}(k^2 \log k)$  bits. This completes the proof of Theorem 3.

The compression result can be lifted to a smaller parameter, namely the number of *conflict triangles*, i.e., triangles incident with at least one edge that is shared by at least three triangles. Recall that nontrivial instances with budget  $k$  have  $\mathcal{O}(k)$  conflict triangles, and observe that having few conflict triangles does not bound the size  $k$  of the desired 2-sphere.

► **Corollary 13** (\*). *The DELETION-TO-2-DIM-SPHERE problem admits a polynomial compression to weighted instances with  $\mathcal{O}(t)$  triangles and bit-size  $\mathcal{O}(t^2)$  where  $t$  is the number of conflict triangles in the input.*

## 6 Conclusion

Our hardness results can be extended easily to cases of finding some other surfaces, such as a torus. Indeed, we can replace in the construction the back sheet with any other shape that has the target topology. Similarly, the positive results can also be extended to the search for small surfaces, again like the torus.

It is clear that the simplicial complex we use to show hardness cannot be embedded in 3-dimensional space. It is unclear how hard the problem 2-DIM-SPHERE is when restricted to simplicial complexes that are embedded in  $\mathbb{R}^3$ . Note that it is not meaningful to parameterize the problem by the dimension of some ambient space because any 2-dimensional simplicial complex can be embedded in  $\mathbb{R}^5$  using the moment curve ([15, Section 1.6]).

A simplicial complex can be generalized to something called, unsurprisingly, a generalized triangulation (or sometimes just referred to as a triangulation). In this setting, we are allowed to identify facets of a common simplex. That is, in a 2-dimensional generalized triangulation we may identify together two distinct edges of the same triangulation. This relaxation can make it harder to even detect a manifold, as there are more cases to consider. Our work here, and related problems, are all still of interest in this setting.

Lastly, the problems discussed in this paper generalize, where possible, in the obvious manner to higher dimensions. In particular, fast detection of 3-sphere subcomplexes (or sub-triangulations) that do not bound a ball are of particular interest for the recognition of the prime decomposition of 3-manifolds.

**Acknowledgments.** This work was initiated during the Fixed-Parameter Computational Geometry Workshop at the Lorentz Center, 2016. We are grateful to the other participants of the workshop and the Lorentz Center for their support.

---

**References**

---

- 1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.
- 2 Bhaskar Bagchi, Benjamin A. Burton, Basudeb Datta, Nitin Singh, and Jonathan Spreer. Efficient algorithms to decide tightness. In Sándor Fekete and Anna Lubiw, editors, *32nd International Symposium on Computational Geometry (SoCG 2016)*, volume 51 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- 3 Nicolas Bonichon, Cyril Gavoille, Nicolas Hanusse, Dominique Poulalhon, and Gilles Schaeffer. Planar graphs, via well-orderly maps and trees. *Graphs and Combinatorics*, 22(2):185–202, 2006. doi:10.1007/s00373-006-0647-2.
- 4 Benjamin A. Burton and Rodney G. Downey. Courcelle’s theorem for triangulations. URL: <http://arxiv.org/abs/1403.2926>.
- 5 Benjamin A. Burton, Clément Maria, and Jonathan Spreer. Algorithms and complexity for Turaev-Viro invariants. In *Automata, Languages, and Programming: 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part 1*, pages 281–293. Springer, 2015. doi:10.1007/978-3-662-47672-7\_23.
- 6 Benjamin A. Burton and William Pettersson. Fixed parameter tractable algorithms in combinatorial topology. In Zhipengand Cai, Alexand Zelikovsky, and Anu Bourgeois, editors, *Computing and Combinatorics: 20th International Conference, COCOON 2014, Atlanta, GA, USA, August 4-6, 2014. Proceedings*, pages 300–311, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-08783-2\_26.
- 7 Oleksiy Busaryev, Sergio Cabello, Chao Chen, Tamal K. Dey, and Yusu Wang. Annotating simplices with a homology basis and its applications. In Fedor V. Fomin and Petteri Kaski, editors, *Algorithm Theory – SWAT 2012 – 13th Scandinavian Symposium and Workshops, Helsinki, Finland, July 4-6, 2012. Proceedings*, volume 7357 of *Lecture Notes in Computer Science*, pages 189–200. Springer, 2012. doi:10.1007/978-3-642-31155-0\_17.
- 8 Chao Chen and Daniel Freedman. Hardness results for homology localization. *Discrete & Computational Geometry*, 45(3):425–448, 2011. doi:10.1007/s00454-010-9322-8.
- 9 Jeff Erickson and Amir Nayyeri. Minimum cuts and shortest non-separating cycles via homology covers. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1166–1176. SIAM, 2011. doi:10.1137/1.9781611973082.88.
- 10 Michael Hartley Freedman. The topology of four-dimensional manifolds. *J. Differential Geom.*, 17(3):357–453, 1982. URL: <http://projecteuclid.org/euclid.jdg/1214437136>.
- 11 Sergei Ivanov. computational complexity (answer). MathOverflow. URL: <http://mathoverflow.net/questions/118357/computational-complexity>.
- 12 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 *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 771–780. IEEE Computer Society, 2008. URL: <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4690923>, doi:10.1109/FOCS.2008.53.
- 13 Clément Maria and Jonathan Spreer. A polynomial time algorithm to compute quantum invariants of 3-manifolds with bounded first Betti number. <http://arxiv.org/abs/1607.02218> [cs.CG], 2016. 15 pages, 3 figures.
- 14 Dániel Marx. A tight lower bound for planar multiway cut with fixed number of terminals. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming – 39th International Colloquium, ICALP 2012*,

## 18:14 The Parameterized Complexity of Finding a 2-Sphere in a Simplicial Complex

- Warwick, UK, July 9-13, 2012, *Proceedings, Part I*, volume 7391 of *Lecture Notes in Computer Science*, pages 677–688. Springer, 2012. doi:10.1007/978-3-642-31594-7\_57.
- 15 Jiří Matoušek. *Using the Borsuk-Ulam Theorem: Lectures on Topological Methods in Combinatorics and Geometry*. Universitext. Springer, 2007.
  - 16 Bojan Mohar. A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM J. Discrete Math.*, 12(1):6–26, 1999. doi:10.1137/S089548019529248X.
  - 17 James R. Munkres. *Elements of Algebraic Topology*. Addison-Wesley, 1993.
  - 18 A. Nabutovsky. Einstein structures: Existence versus uniqueness. *Geometric & Functional Analysis GAFA*, 5(1):76–91, 1995. doi:10.1007/BF01928216.
  - 19 Saul Schleimer. Sphere recognition lies in NP. In Michael Usher, editor, *Low-Dimensional and Symplectic Topology*, volume 82 of *Proceedings of Symposia in Pure Mathematics*, pages 183–213. AMS, 2011.
  - 20 W. T. Tutte. A census of planar triangulations. *Canad. J. Math.*, 14:21–38, 1962. doi:10.4153/CJM-1962-002-9.

# On Long Words Avoiding Zimin Patterns

Arnaud Carayol<sup>1</sup> and Stefan Göller<sup>\*2</sup>

- 1 Université Paris-Est, LIGM (UMR 8049), CNRS, ENPC, ESIEE, UPEM, Marne-la-Vallée, France  
carayol@u-pem.fr
- 2 LSV, CNRS & ENS Cachan, Université Paris-Saclay, Paris, France  
goeller@lsv.fr

---

## Abstract

A pattern is encountered in a word if some infix of the word is the image of the pattern under some non-erasing morphism. A pattern  $p$  is unavoidable if, over every finite alphabet, every sufficiently long word encounters  $p$ . A theorem by Zimin and independently by Bean, Ehrenfeucht and McNulty states that a pattern over  $n$  distinct variables is unavoidable if, and only if,  $p$  itself is encountered in the  $n$ -th Zimin pattern. Given an alphabet size  $k$ , we study the minimal length  $f(n, k)$  such that every word of length  $f(n, k)$  encounters the  $n$ -th Zimin pattern. It is known that  $f$  is upper-bounded by a tower of exponentials. Our main result states that  $f(n, k)$  is lower-bounded by a tower of  $n - 3$  exponentials, even for  $k = 2$ . To the best of our knowledge, this improves upon a previously best-known doubly-exponential lower bound. As a further result, we prove a doubly-exponential upper bound for encountering Zimin patterns in the abelian sense.

**1998 ACM Subject Classification** G.2.1 Combinatorics, F.4.3. Formal Languages

**Keywords and phrases** Unavoidable patterns, combinatorics on words, lower bounds

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.19

## 1 Introduction

A pattern is a finite word over some set of pattern variables. A pattern matches a word if the word can be obtained by substituting each variable appearing in the pattern by a non-empty word. The pattern  $xx$  matches the word  $nana$  when  $x$  is replaced by the word  $na$ . A word encounters a pattern if the pattern matches some infix of the word. For example, the word  $banana$  encounters the pattern  $xx$  (as the word  $nana$  is one of its infixes). The pattern  $xyx$  is encountered in precisely those words that contain two non-consecutive occurrences of the same letter, as e.g., the word  $abca$ .

A pattern is unavoidable if over every finite alphabet every sufficiently long word encounters the pattern. Equivalently, by König's Lemma, a pattern is unavoidable if over every finite alphabet all infinite words encounter the pattern. If it is not the case, the pattern is said to be avoidable. The pattern  $xyx$  is easily seen to be unavoidable since every sufficiently long word over a non-empty finite alphabet must contain two non-consecutive occurrences of the same letter. On the other hand, the pattern  $xx$  is avoidable as Thue [19] gave an infinite word over a ternary alphabet that does not encounter the pattern  $xx$ .

A precise characterization of unavoidable patterns was found by Zimin [20] and independently by Bean, Ehrenfeucht and McNulty [6], see also [13] for a more recent proof. This

---

\* Supported by Labex Digicosme, Univ. Paris-Saclay, project VERICONISS.



© Arnaud Carayol and Stefan Göller;

licensed under Creative Commons License CC-BY

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 19; pp. 19:1–19:13

Leibniz International Proceedings in Informatics



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

characterization is based on a family  $(Z_n)_{n \geq 0}$  of unavoidable patterns, called the Zimin patterns. The Zimin patterns over the pattern variables  $\{x_1, x_2, \dots\}$  are defined by  $Z_0 = \varepsilon$  and  $Z_{n+1} = Z_n x_{n+1} Z_n$  for all  $n \geq 0$ . A pattern over  $n$  distinct pattern variables is unavoidable if, and only if, the pattern itself is encountered in the  $n$ -th Zimin pattern  $Z_n$ . Zimin patterns can therefore be viewed as the canonical patterns for unavoidability.

Due to the canonical status of Zimin patterns, it is natural to investigate the smallest word length  $f(n, k)$  that guarantees that every word over a  $k$ -letter alphabet of this length encounters the  $n$ -th pattern  $Z_n$ . Computing the exact value of  $f(n, k)$  for  $n \geq 1$  and  $k \geq 2$ , or at least giving upper and lower bounds on its value, has been the topic of several articles in recent years [2, 18, 12, 3]. For small values of  $n$  and  $k$ , known results from [12, 11] are summarized in the following table, taken from [12] and enriched with results from [11].

	2	3	4	5	$k$
1	1	1	1	1	1
2	5	7	9	11	$2k + 1$
3	29	$\leq 319$	$\leq 3169$	$\leq 37991$	$\leq \sqrt{e} 2^k (k + 1)! + 2k + 1$
4	$\in [10483, 236489]$				
$n$					

In general, Cooper and Rorabaugh [2, Theorem 1.1] showed that the value of  $f(n, k)$  is upper-bounded by a tower of exponentials of height  $n - 1$ . To make this more precise let us define the tower function  $\text{Tower} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  inductively as follows:  $\text{Tower}(0, k) = 1$  and  $\text{Tower}(n + 1, k) = k^{\text{Tower}(n, k)}$  for all  $n, k \in \mathbb{N}$ .

► **Theorem 1** (Cooper/Rorabaugh [2]). *For all  $n \geq 1$  and  $k \geq 2$ ,  $f(n, k) \leq \text{Tower}(n - 1, K)$ , where  $K = 2k + 1$ .*

In stark contrast with this upper bound, Cooper and Rorabaugh showed that  $f(n, k)$  is lower-bounded doubly-exponentially in  $n$  for every fixed  $k \geq 2$ . To our knowledge, this is the best known lower bound for  $f$ .

► **Theorem 2** (Cooper/Rorabaugh [2]).  *$f(n, k) \geq k^{2^{n-1}(1+o(1))}$ .*

This lower bound is obtained by estimating the expected number of occurrences of  $Z_n$  in long words over a  $k$ -letter alphabet using the first moment method.

In the conclusion, we address the limitation of this method to show non-elementary lower bounds for  $f$ .

**Our contributions.** Our main contribution is to prove a lower bound for  $f(n, k)$  that is non-elementary in  $n$  even for  $k = 2$ . We use Stockmeyer’s yardstick construction [17] to construct for each  $n \geq 1$ , a family of words of length at least  $\text{Tower}(n - 1, 2)$  (that we call higher-order counters here). We then show that a counter of order  $n$  does not encounter  $Z_n$  (for  $n \geq 3$ ). As these words are over an alphabet of size  $2n - 1$ , this immediately establishes that  $f(n, 2n - 1) \geq \text{Tower}(n - 1, 2)$  (cf. Corollary 12).

Stockmeyer’s yardstick construction is a well-known technique to prove non-elementary lower bounds in computer science, for instance it is used to show that the first-order theory of binary words with order, is non-elementary, see for instance [10] for a proof.

By using a carefully chosen encoding we are able to prove a lower bound for  $f$  over a binary alphabet. Namely for all  $n \geq 4$ , it holds  $f(n, 2) \geq \text{Tower}(n - 3, 2)$  (cf. Corollary 14).

As a spin-off result, we also consider the abelian setting. Matching a pattern in the abelian sense is a weaker condition, where one only requires that all infixes that are matching



a pattern variable must have the same number of occurrences of each letter (instead of being the same words). This gives rise to the notions of avoidability and unavoidability of patterns in the abelian sense. Every pattern that is unavoidable is in particular unavoidable in the abelian sense. However, the converse does not hold in general as witnessed by the pattern  $xyzxyxuxyxyzyx$ , as shown in [5]. Even though Zimin patterns lose their canonical status in the abelian setting, the function  $g(n, k)$ , which is an abelian analog of the function  $f(n, k)$ , has been studied [18]. For this function, Tao [18] establishes a lower bound that turns out to be doubly-exponential from the estimations in [9]. The upper bound is inherited from the non-abelian setting and is hence non-elementary. We improve this upper bound to be doubly-exponential (Theorem 22). We also provide a simple proof using the first moment method that  $g$  admits a doubly-exponential lower bound which does not require the elaborate estimations of [9].

**Connection to the equivalence problem of deterministic pushdown automata.** The equivalence problem for deterministic pushdown automata (dpda) is a famous problem in theoretical computer science. Its decidability has been established by Sénizergues in 1997 and Stirling proved in 2001 the first complexity-theoretic upper bound, namely a tower of exponentials of elementary height [16] (in  $\mathbf{F}_3$  in terms of Schmitz' classification [14]), see also [8] for a more recent presentation.

In [15, p. 24, C1] Sénizergues outlines a link between the complexity of Stirling's algorithm and the function  $f(n, k)$  and remarks that  $f(n, k)$  seems to be non-elementary in  $n$  for all fixed  $k \geq 2$ . The present article substantiates this remark.

**Organization of the paper.** We introduce necessary notations in Section 2. We show that  $f(n, 2n - 1) \geq \text{Tower}(n - 1, 2)$  in Section 3. We lift this result to unavoidability over a binary alphabet in Section 4, where we show that  $f(n, 2) \geq \text{Tower}(n - 3, 2)$  for all  $n \geq 4$ . Our doubly-exponential bounds on abelian avoidability are presented in Section 5. We conclude in Section 6.

## 2 Preliminaries

For every two integers  $i, j$ , we write  $[i, j]$  for the set  $\{i, i + 1, \dots, j\}$  and  $[j]$  for  $\{1, \dots, k\}$ . By  $\mathbb{N}$  we denote the non-negative integers and by  $\mathbb{N}^+$  the positive integers.

If  $A$  is a finite set of symbols, we denote by  $A^*$  the set of all words over  $A$  and by  $A^+$  the set of all non-empty words over  $A$ . We write  $\varepsilon$  for the empty word. For a word  $u \in A^*$ , we denote by  $|u|$  its length. For two words  $u$  and  $v$ , we denote by  $u \cdot v$  (or simply  $uv$ ) their concatenation. A word  $v$  is a *prefix* of a word  $u$ , denoted by  $v \sqsubseteq u$ , if there exists a word  $z$  such that  $u = vz$ . If  $z$  is non-empty, we say that  $v$  is a *strict prefix*<sup>1</sup> of  $u$ . A word  $v$  is a suffix of a word  $u$  if there exist a word  $z$  such that  $u = zv$ . If  $z$  is non-empty, we say that  $v$  is a *strict suffix* of  $u$ . A word  $v$  is an infix of a word  $u$  if there exists  $z_1$  and  $z_2$  such that  $u = z_1vz_2$ . If both  $z_1$  and  $z_2$  are non-empty,  $v$  is a *strict infix* of  $u$ . If  $v$  is an infix  $u$  and  $u$  can be written as  $z_1uz_2$ , the integer  $|z_1|$  is called an *occurrence* of  $v$  in  $u$ . For  $a \in A$ , we denote by  $|u|_a$  the number of occurrences of the symbol  $a$  in  $u$ .

Given two non-empty sets  $A$  and  $B$ , a *morphism* is a function  $\phi : A^* \rightarrow B^*$  that satisfies  $\phi(a_1a_2) = \phi(a_1)\phi(a_2)$  for all  $a_1, a_2 \in A$ . Thus, every morphism can simply be written as a

<sup>1</sup> Our definition of strict prefix is slightly non-standard as  $\varepsilon$  is a strict prefix of any non-empty word.

## 19:4 On Long Words Avoiding Zimin Patterns

function from  $A$  to  $B^*$ . A morphism  $\phi$  is said to be *non-erasing* if  $\phi(a) \neq \varepsilon$  for all  $a \in A$  and  $\phi$  is *alphabetic* if  $\psi(a) \in B$  for all  $a \in A$ .

Let us fix a countable set  $\mathcal{X} = \{x_1, x_2, \dots\}$  of *pattern variables*. A *pattern* is a finite word over  $\mathcal{X}$ . Let  $\rho = \rho_1 \cdots \rho_n$  be a pattern of length  $n$ . A finite or infinite word  $w$  *matches*  $\rho$  if  $w = \psi(\rho)$  for some non-erasing morphism  $\psi$ . A finite or infinite word  $w$  *encounters*  $\rho$  if some infix of  $w$  matches  $\rho$ . A pattern  $\rho$  is said to be *unavoidable* if for all  $k \geq 1$  all but finitely many finite words (equivalently every infinite word, by König's Lemma) over the alphabet  $[k]$  encounter  $\rho$ . Otherwise we say  $\rho$  is *avoidable*. Unavoidable patterns are characterized by the so-called Zimin patterns. For all  $n \geq 0$ , the  $n$ -th *Zimin pattern*  $Z_n$  is given by:

$$Z_0 = \varepsilon \quad \text{and} \quad Z_{n+1} = Z_n x_{n+1} Z_n \quad \text{for all } n \geq 0.$$

For instance, we have  $Z_1 = x_1$ ,  $Z_2 = x_1 x_2 x_1$  and  $Z_3 = x_1 x_2 x_1 x_3 x_1 x_2 x_1$ .

The following theorem gives a decidable characterization of unavoidable patterns.

► **Theorem 3** (Bean/Ehrenfeucht/McNulty [6], Zimin [20]). *A pattern  $\rho$  containing  $n$  different variables is unavoidable if, and only if,  $Z_n$  encounters  $\rho$ .*

For instance, the pattern  $x_1 x_2 x_1 x_2$  is avoidable because it matches  $x_1 x_1$  which itself is not encountered in  $Z_n$  for all  $n \in \mathbb{N}$ . This characterization justifies the study of the following Ramsey-like function.

► **Definition 4.** Let  $n, k \geq 1$ . We define  $f(n, k) = \min\{\ell \geq 1 \mid \forall w \in [k]^\ell : w \text{ encounters } Z_n\}$ .

As we mainly work with Zimin patterns, we introduce the notions of Zimin type (i.e. the maximal Zimin pattern that matches a word) and Zimin index (i.e. the maximal Zimin pattern that a word encounters) and their basic properties.

The *Zimin type*  $Z\text{Type}(w)$  of a word  $w$  is the largest  $n$  such that  $w = \varphi(Z_n)$  for some non-erasing morphism  $\varphi$ . For instance, we have  $Z\text{Type}(aaab) = 1$ ,  $Z\text{Type}(aba) = 2$  and  $Z\text{Type}(a^7 b a^7) = 4$ . Note that the Zimin type of any non-empty word is greater or equal to 1 and the Zimin type of the empty word is 0.

Following the definition of the Zimin patterns, the Zimin type of a word can be inductively characterized as follows:

► **Fact 5.** *For any non-empty word  $w$ ,  $Z\text{Type}(w) = 1 + \max\{Z\text{Type}(\alpha) \mid w = \alpha\beta\alpha : \alpha, \beta \neq \varepsilon\}$ , with the convention that the maximum of the empty set is 0.*

► **Definition 6.** The *Zimin index*  $Z\text{imin}(w)$  of a non-empty word  $w$  is the maximum Zimin type of an infix of  $w$ .

For instance, we have  $Z\text{imin}(aaab) = 2$  and  $Z\text{imin}(bbaba) = 2$ . As a further example note that  $Z\text{imin}(baaabaaa) = 3$  although  $Z\text{Type}(baaabaaa) = 1$ .

► **Lemma 7.** *For any word  $w$ , we have the following properties:*

- $Z\text{Type}(w) \leq Z\text{imin}(w)$ ,
- for any infix  $w'$  of  $w$ ,  $Z\text{imin}(w') \leq Z\text{imin}(w)$ ,
- $Z\text{imin}(w) \leq \lfloor \log_2(|w| + 1) \rfloor$ .

**Proof.** The first two points directly follow from the definition. For the last point, note that for a word  $w$  to encounter the  $n$ -th Zimin pattern  $Z_n$ , it must be of length at least  $|Z_n|$ . As  $Z_n$  has length  $2^n - 1$ , we have  $2^{Z\text{imin}(w)} - 1 \leq |w|$ , which implies the announced bound. ◀

### 3 The Zimin index of higher-order counters

In this section we show that there is a family of words, that we refer to as “higher-order counters”, whose lengths are non-elementary in  $n$  and whose Zimin index is  $n - 1$ , allowing us to show that  $f(2n - 1, n) \geq \text{Tower}(n - 1, 2)$ . In Section 3.1 we introduce higher-order counters and in Section 3.2 we show that their Zimin index is precisely  $n - 1$  including the mentioned lower bound on  $f$ .

#### 3.1 Higher-order counters à la Stockmeyer

In this section we introduce counters that encode values ranging from 0 to a tower of exponentials. To the best of our knowledge this construction was introduced by Stockmeyer to show non-elementary complexity lower bounds and is often referred to as the “yardstick construction” [17]. We refer to such counters as “higher-order counters” in the following.

To make the notation less cluttered, we define  $\tau : \mathbb{N} \rightarrow \mathbb{N}$ , the *tower of twos function* which satisfies  $\tau(n) = \text{Tower}(n, 2)$  for all  $n \geq 0$ . For all  $n \geq 1$ , we define an alphabet  $\Sigma_n$  by taking  $\Sigma_1 = \{0_1, 1_1\}$  and for all  $n > 1$ ,  $\Sigma_n = \Sigma_{n-1} \cup \{0_n, 1_n\}$ . We say the symbols  $0_n$  and  $1_n$  have *order*  $n$ . We define  $\Sigma = \cup_{n \geq 1} \Sigma_n$  to be set of all these symbols.

For all  $n \geq 1$  and for all  $i \in [0, \tau(n) - 1]$ , we define a word over  $\Sigma_n$  called *the  $i$ -th counter of order  $n$*  and denoted by  $\llbracket i \rrbracket_n$ . The definition proceeds by induction on  $n$ . For  $n = 1$ , there are only two counters  $\llbracket 0 \rrbracket_1$  and  $\llbracket 1 \rrbracket_1$  (recall that  $\tau(1) = 2$ ). We define  $\llbracket 0 \rrbracket_1 = 0_1$  and  $\llbracket 1 \rrbracket_1 = 1_1$  and for  $n \geq 1$  and  $i \in [0, \tau(n+1) - 1]$  we define

$$\llbracket i \rrbracket_{n+1} = \llbracket 0 \rrbracket_n b_0 \llbracket 1 \rrbracket_n b_1 \cdots \llbracket \tau(n) - 1 \rrbracket_n b_{\tau(n)-1},$$

where  $b_0 b_1 \cdots b_{\tau(n)-2} b_{\tau(n)-1}$  is the binary decomposition of  $i$  over the alphabet  $\{0_{n+1}, 1_{n+1}\}$  with  $b_0$  the least significant bit (i.e.  $i = \sum_{j=0}^{\tau(n)-1} \bar{b}_j \cdot 2^j$  where  $\bar{b}_j = 0$  if  $b_j = 0_{n+1}$  and  $\bar{b}_j = 1$  if  $b_j = 1_{n+1}$ ).

For  $\llbracket 11 \rrbracket_3$ , we have  $11 = 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3$  and hence

$$\llbracket 11 \rrbracket_3 = \underbrace{0_1 0_2 1_1 0_2}_{\llbracket 0 \rrbracket_2} \mathbf{1}_3 \underbrace{0_1 1_2 1_1 0_2}_{\llbracket 1 \rrbracket_2} \mathbf{1}_3 \underbrace{0_1 0_2 1_1 1_2}_{\llbracket 2 \rrbracket_2} \mathbf{0}_3 \underbrace{0_1 1_2 1_1 1_2}_{\llbracket 3 \rrbracket_2} \mathbf{1}_3.$$

The following lemma can easily be proven by induction on  $n$ .

► **Lemma 8.** *Let  $n \geq 1$ .*

1. *For all  $i \neq j \in [0, \tau(n) - 1]$  we have  $\llbracket i \rrbracket_n \neq \llbracket j \rrbracket_n$ .*
2. *If  $n > 1$ , then for all  $i \in [0, \tau(n) - 1]$  and  $j \in [0, \tau(n-1) - 1]$  the counter  $\llbracket j \rrbracket_{n-1}$  has exactly one occurrence in  $\llbracket i \rrbracket_n$ .*

The length  $L_n$  of an order- $n$  counter satisfies the following equations:  $L_1 = 1$  and  $L_{n+1} = \tau(n) \cdot (L_n + 1)$  for  $n \geq 1$ . Note that in particular we have  $L_n \geq \tau(n - 1)$  for all  $n \geq 1$ .

#### 3.2 Higher-order counters have small Zimin index

The aim of this section is to give an upper bound on the Zimin index of counters of order  $n$ . A first simple remark is that the Zimin index of any counter of order  $n$  is bounded by the Ziminindex of  $\llbracket 0 \rrbracket_n$ .

► **Lemma 9.** *For all  $n \geq 1$  and for all  $i \in [0, \tau(n) - 1]$ ,  $\text{Zimin}(\llbracket i \rrbracket_n) \leq \text{Zimin}(\llbracket 0 \rrbracket_n)$ .*

## 19:6 On Long Words Avoiding Zimin Patterns

**Proof Sketch.** Let  $n \geq 1$  and  $i \in [0, \tau(n) - 1]$ . Consider the morphism  $\varphi$  defined by  $\varphi(0_n) = \varphi(1_n) = 0_n$  and  $\varphi(x) = x$  for  $x \in \Sigma_{n-1}$ . We have  $\varphi(\llbracket i \rrbracket_n) = \llbracket 0 \rrbracket_n$ . Moreover as  $\varphi$  is alphabetic, if an infix  $\alpha$  of  $\llbracket i \rrbracket_n$  matches  $Z_\ell$  for some  $\ell \geq 0$  then  $\varphi(\alpha)$  is an infix of  $\varphi(\llbracket i \rrbracket_n) = \llbracket 0 \rrbracket_n$  that also matches  $Z_\ell$ . The inequality claimed follows.  $\blacktriangleleft$

This leads us to the main result of this section.

**► Theorem 10.** *For all  $n \geq 3$ ,  $\text{Zimin}(\llbracket 0 \rrbracket_n) \leq n - 1$ .*

**Proof.** The proof proceeds by induction on  $n \geq 3$ . The base case can be checked using a computer program.

Assume that the property holds for some  $n \geq 3$ . Let us show that  $\text{Zimin}(\llbracket 0 \rrbracket_{n+1}) \leq n$ . Let  $\alpha\beta\alpha$  be an infix of  $\llbracket 0 \rrbracket_{n+1}$  for some non-empty words  $\alpha$  and  $\beta$ . It is enough to show that  $\text{ZType}(\alpha) \leq n - 1$ . We distinguish the following cases depending on the number occurrences of  $0_{n+1}$  in  $\alpha$ .

**Case 1:  $\alpha$  contains no occurrences of  $0_{n+1}$ .** Then  $\alpha$  is an infix of some  $\llbracket i \rrbracket_n$  for some  $i \in [0, \tau(n) - 1]$ . Using Lemma 9 and the induction hypothesis, we have  $\text{ZType}(\alpha) \leq \text{Zimin}(\alpha) \leq \text{Zimin}(\llbracket i \rrbracket_n) \leq \text{Zimin}(\llbracket 0 \rrbracket_n) \leq n - 1$ .

**Case 2:  $\alpha$  contains at least two occurrences of  $0_{n+1}$ .** By definition of counters,  $\alpha$  has an infix of the form  $0_{n+1}\llbracket i \rrbracket_n 0_{n+1}$  for some  $i \in [0, \tau(n) - 1]$ . Hence  $\llbracket 0 \rrbracket_{n+1}$  would contain two occurrences of  $0_{n+1}\llbracket i \rrbracket_n 0_{n+1}$  which is not possible (cf. Lemma 7).

**Case 3:  $\alpha$  contains exactly one occurrence of  $0_{n+1}$ .** By definition of  $\llbracket 0 \rrbracket_{n+1}$ , there exists  $i \neq j \in [0, \tau(n) - 1]$  such that  $\alpha$  is of the form  $u0_{n+1}v$  with  $u$  a suffix of both  $\llbracket i \rrbracket_n$  and  $\llbracket j \rrbracket_n$  and  $v$  a prefix of both  $\llbracket i + 1 \rrbracket_n$  and  $\llbracket j + 1 \rrbracket_n$ .

Consider the morphism  $\psi$  that erases all symbols in  $\Sigma_{n-1}$  and replaces  $0_n$  and  $1_n$  by 0 and 1, respectively. Let us assume that

$$\psi(u) = b_{\tau(n-1)-\ell_0} \cdots b_{\tau(n-1)-1} \quad \text{and} \quad \psi(v) = c_0 \cdots c_{\ell_1-1}$$

for some  $\ell_0 \in [0, \tau(n-1)]$  and  $\ell_1 \in [0, \tau(n-1)]$  and  $b_k \in \{0, 1\}$  for all  $k \in [\tau(n-1) - \ell_0, \tau(n-1) - 1]$  and  $c_k \in \{0, 1\}$  for all  $k \in [0, \ell_1 - 1]$ .

Let us start by showing that  $\ell_0 + \ell_1 < \tau(n-1)$ .

By definition of counters,  $b_{\tau(n-1)-1}$  is the most significant bit of the binary representation (of length  $\tau(n-1)$ ) of  $i$  and  $j$  and  $c_0$  is the least significant bit of the binary representation of both  $i + 1$  and  $j + 1$ . More formally, there exist  $x_i$  and  $x_j \in [0, 2^{\tau(n-1)-\ell_0} - 1]$  and  $y_i$  and  $y_j \in [0, 2^{\tau(n-1)-\ell_1} - 1]$  such that

$$i = x_i + 2^{\tau(n-1)-\ell_0} \cdot B \quad j = x_j + 2^{\tau(n-1)-\ell_0} \cdot B \quad i + 1 = C + 2^{\ell_1} y_i \quad j + 1 = C + 2^{\ell_1} y_j$$

$$\text{with } B = \sum_{k=0}^{\ell_0-1} b_{\tau(n-1)-\ell_0-k} \cdot 2^k \quad \text{and} \quad C = \sum_{k=0}^{\ell_1-1} c_k \cdot 2^k.$$

Assume by way of contradiction that  $\ell_0 + \ell_1 \geq \tau(n-1)$ . In particular, this implies  $2^{\ell_1} \geq 2^{\tau(n-1)-\ell_0}$ . And hence,

$$\begin{aligned} x_i &= i \pmod{2^{\tau(n-1)-\ell_0}} && \text{by definition of } i \\ &= C - 1 + 2^{\ell_1} y_i \pmod{2^{\tau(n-1)-\ell_0}} \\ &= C - 1 \pmod{2^{\tau(n-1)-\ell_0}} && \text{as } 2^{\tau(n-1)-\ell_0} \text{ divides } 2^{\ell_1}. \end{aligned}$$

A similar reasoning shows that  $x_j = C - 1 \pmod{2^{\tau(n-1)-\ell_0}}$ . Hence  $x_i = x_j$  and hence  $i = j$  which brings the contradiction.

As  $\ell_0 + \ell_1 < \tau(n-1)$ , there exists some  $i_0 \in [0, \tau(n) - 1]$  such that  $v$  is a prefix and  $u$  is a suffix of  $\llbracket i_0 \rrbracket_n$ . That is, the binary representation of  $i_0$  of length  $\tau(n-1)$  has  $c_0 \cdots c_{\ell_1-1}$  as  $\ell_1$  least significant bits and  $b_{\tau(n-1)-\ell_0} \cdots b_{\tau(n-1)-1}$  as  $\ell_0$  most significant bits. In particular, as  $\ell_0 + \ell_1 < \tau(n-1)$ , we have that  $\llbracket i_0 \rrbracket_n = vru$  for some non-empty  $r$ .

We claim that  $Z\text{Type}(\alpha) \leq Z\text{imin}(\llbracket i_0 \rrbracket_n)$  by which we would be done since then  $Z\text{Type}(\alpha) \leq Z\text{imin}(\llbracket i_0 \rrbracket_n) \leq Z\text{imin}(\llbracket 0 \rrbracket_n) \leq n-1$  by Lemma 9 and the induction hypothesis.

Assume that  $\alpha$  can be written as  $\gamma\delta\gamma$  for non-empty  $\gamma$  and  $\delta$ . Using Fact 5, it is enough to show that  $Z\text{Type}(\gamma) + 1 \leq Z\text{imin}(\llbracket i_0 \rrbracket_n)$ . Recall that  $\alpha = u0_{n+1}v$  and  $\alpha$  contains only one occurrence of  $0_{n+1}$ . It follows that  $\gamma$  must be a prefix of  $u$  and a suffix of  $v$ . In particular,  $\llbracket i_0 \rrbracket_n = vru$  contains  $\gamma r \gamma$  as an infix. And hence, we have  $Z\text{Type}(\gamma) + 1 \leq Z\text{Type}(\gamma r \gamma) \leq Z\text{imin}(\llbracket i_0 \rrbracket_n)$ . ◀

The upper bound on the Zimin index of higher-order counters established in the previous theorem is tight.

► **Theorem 11.** *For all  $n \geq 3$  and  $i \in [0, \tau(n) - 1]$ ,  $Z\text{imin}(\llbracket i \rrbracket_n) = n - 1$ .*

For  $n \geq 3$ , the counter  $\llbracket 0 \rrbracket_n$  over the alphabet  $\{0_1, 1_1, \dots, 0_{n-1}, 1_{n-1}, 0_n\}$  of size  $2n-1$  has Zimin index at most  $n-1$ . In particular,  $\llbracket 0 \rrbracket_n$  does not encounter the pattern  $Z_n$ . Therefore its length  $L_n \geq \tau(n-1)$  gives a lower bound for the value of  $f(n, 2n-1)$ .

► **Corollary 12.** *For all  $n \geq 3$ ,  $f(n, 2n-1) \geq \tau(n-1) = \text{Tower}(n-1, 2)$ .*

## 4 Reduction to the binary alphabet

In this section, we show how to encode a higher-order counter seen in Section 3 over the binary alphabet  $\{0, 1\}$  while still preserving a relatively low Zimin index. For this we apply to higher-order counters, the morphism  $\psi$  defined for all  $n \geq 1$ , as follows

$$\psi(0_n) = 00(01)^{n-1}00 \quad \psi(1_n) = 11(01)^{n-1}11.$$

For all  $n \geq 1$  and  $i \in [0, \tau(n) - 1]$ , we define  $\{\!\{ i \}\!\}_n = \psi(\llbracket i \rrbracket_n)$ .

The set of images of the letters in  $\Sigma$  by this morphism forms what is known as an infix code, i.e.  $\psi(a)$  is not an infix of  $\psi(b)$  for any two letters  $a, b \in \Sigma$  with  $a \neq b$ . In addition to being an infix code, the morphism was designed so that we are able to attribute a non-ambiguous partial decoding to most infixes of an encoded word (cf. Lemma 17) and the encoding of  $0_n$  and  $1_n$  differ by their first and last symbol.

Applying a non-erasing morphism to a word can only increase its Zimin index. We will see in the remainder of this section that the Zimin index of higher-order counters is increased by at most 2 when the morphism  $\psi$  is applied. It is possible that another choice of morphism would bring a better upper bound. However, note that the proof we present is tightly linked to the above-mentioned properties of  $\psi$  that are decisive for the proof to work.

This section is devoted to establishing the following theorem.

► **Theorem 13.** *For all  $n \geq 2$  and for all  $i \in [0, \tau(n) - 1]$ ,  $Z\text{imin}(\{\!\{ i \}\!\}_n) \leq n + 1$ .*

Recalling that an order- $n$  counter has length at least  $\tau(n-1)$  and that applying  $\psi$  can only increase the length, we immediately obtain from Theorem 13 a non-elementary lower bound for  $f(n, 2)$  whenever  $n \geq 4$ .

► **Corollary 14.** For all  $n \geq 4$ ,  $f(n, 2) \geq \tau(n - 3) = \text{Tower}(n - 3, 2)$ .

The proof of Theorem 13 which is essentially a reduction to the proof of Theorem 10 is given in Section 4.2. To perform this reduction, we first establish basic properties of the morphism  $\psi$  and its decoding in Section 4.1.

#### 4.1 Parsing the code $\psi$

A word  $w \in \{0, 1\}^*$  is *coded* by  $\psi$  (or simply a *coded word*) if it is the image by  $\psi$  of some word  $v$  over  $\Sigma^*$ . As the image of  $\psi$  is an infix code, the word  $v \in \Sigma^*$  is unique. However in our proof, we need to take into consideration all infixes of a coded word. To be able to reuse the proof techniques of Theorem 10, it is necessary to associate to an infix of a coded word a partial decoding called a *parse*.

Let us therefore consider the following sets,  $C = \psi(\Sigma) = \{\psi(0_k) \mid k \geq 1\} \cup \{\psi(1_k) \mid k \geq 1\}$  the image of the morphism,  $L = \{v \in \{0, 1\}^* \mid \exists u \in \{0, 1\}^+, uv \in C\}$  the set of strict suffixes of  $C$ ,  $R = \{u \in \{0, 1\}^* \mid \exists v \in \{0, 1\}^+, uv \in C\}$  the set of strict prefixes of  $C$  and  $F = \{v \in \{0, 1\}^* \mid \exists u, w \in \{0, 1\}^+, uvw \in C\}$  the set of strict infixes of  $C$ .

It is easy to see that every infix of a coded word belongs to  $F \cup LC^*R$ . This leads us to define a *parse*  $p$  as a triple  $(\ell, u, r)$  in  $L \times \Sigma^* \times R$ . The word  $u$  will be called the *center* of the parse  $p$ . The *value* of the parse  $(\ell, u, r)$  is the word  $\ell\psi(u)r \in \{0, 1\}^*$ . We say that  $\alpha$  admits a parse  $p$  if  $\alpha$  is the value of  $p$ .

By the above fact, for all coded words all of its infixes not belonging to  $F$  have at least one parse. However, the parse is not necessarily unique. For instance, consider the infix  $\alpha = 0000000000 = 0^{10}$  which appears in  $\psi(0_10_10_1)$ . It can be parsed as  $(\varepsilon, 0_10_1, 00)$ ,  $(0, 0_10_1, 0)$  and as  $(00, 0_10_1, \varepsilon)$ . However, we will provide sufficient conditions on an infix to admit a unique parse.

► **Definition 15.** A word  $\alpha \in \{0, 1\}^*$  is *simple* if either  $|\alpha| < 11$ , or  $\alpha$  belongs to  $F$  or  $0^{10}$  is an infix of  $\alpha$  or  $1^{10}$  is an infix of  $\alpha$ .

This definition will be justified by the fact that for non-simple infixes there is exactly one possible parse. Moreover the term simple is justified in the context of this proof by the fact that simple infixes of  $\{\{i\}_n\}$  can be shown to have Zimin index at most  $n - 1$  for all  $n \geq 4$  and all  $i \in [0, \tau(n) - 1]$ .

► **Lemma 16.** For all  $n \geq 4$ , for all  $i \in [0, \tau(n) - 1]$ , all simple infixes of  $\{\{i\}_n\}$  have Zimin index at most  $n - 1$ .

► **Lemma 17.** Any non-simple infix of a coded word admits a unique parse.

Thus, we will refer to the unique parse of a non-simple infix  $\alpha$  of a coded word as *the parse of  $\alpha$* .

The notion of occurrence naturally extends to parses. Let  $w = w_0 \cdots w_{|w|-1} \in \Sigma^*$  and  $p = (\ell, u = u_0 \cdots u_{|u|-1}, r)$  be a parse, an *occurrence of  $p$  in  $w$*  is an occurrence  $m$  of  $u$  in  $w$  such that whenever  $\ell$  is non-empty we have  $m \neq 0$  and  $\ell$  is suffix of  $\psi(w_{m-1})$  and similarly whenever  $r$  is non-empty we have  $m + |u| < |w|$  and  $r$  is a prefix of  $\psi(w_{m+|u|})$ . For a word  $w \in \Sigma^*$  and a non-simple infix  $\alpha$  of  $\psi(w)$ , there is a one-to-one correspondence between the occurrences of  $\alpha$  in  $\psi(w)$  and the occurrences of its parse  $p_\alpha$  in  $\psi(w)$ .

► **Definition 18.** For an occurrence  $m$  of a parse  $p = (\ell, u, r)$  in  $w$ , we define its context as the word in  $\Sigma^*$  equal to  $w[m - \delta_0, m + |u| + \delta_1]$ , where  $\delta_0 = 0$  if  $\ell = \varepsilon$  and  $\delta_0 = 1$  otherwise and  $\delta_1 = 0$  if  $r = \varepsilon$  and  $\delta_1 = 1$  otherwise.

By definition the context  $c$  of some occurrence of a parse  $p = (\ell, u, r)$  in  $w$  is an infix of  $w$  containing  $u$  as an infix. Moreover the value  $\alpha$  of  $p$  is an infix of  $\psi(c)$ .

## 4.2 Upper bound on the Zimin index

To establish Theorem 13, we prove the following stronger statement.

► **Theorem 19.** *For all  $n \geq 2$  and for all  $i \in [0, \tau(n) - 1]$ ,*

$$\begin{aligned} \text{Zimin}(\{\!\{ i \}\!\}_n \psi(0_{n+1})) &\leq n + 1, & \text{Zimin}(\{\!\{ i \}\!\}_n \psi(1_{n+1})) &\leq n + 1, \\ \text{Zimin}(\psi(0_{n+1}) \{\!\{ i \}\!\}_n) &\leq n + 1, & \text{Zimin}(\psi(1_{n+1}) \{\!\{ i \}\!\}_n) &\leq n + 1. \end{aligned}$$

**Proof Sketch.** We proceed by induction on  $n$ . For the case  $n = 2$  and  $n = 3$ , the property is checked using a computer program. For the induction step assume that the property holds for some  $n \geq 3$  and let us show that it holds for  $n + 1$ . Let  $i \in [0, \tau(n + 1) - 1]$ , we will only show that  $\text{Zimin}(\{\!\{ i \}\!\}_{n+1}) \leq n + 2$ . The upper bound on  $\text{Zimin}(\{\!\{ i \}\!\}_{n+1}, \psi(0_{n+2}))$ ,  $\text{Zimin}(\{\!\{ i \}\!\}_{n+1}, \psi(1_{n+2}))$ ,  $\text{Zimin}(\psi(0_{n+2}) \{\!\{ i \}\!\}_{n+1})$  and  $\text{Zimin}(\psi(1_{n+2}) \{\!\{ i \}\!\}_{n+1})$  can be deduced from this, but still require a tedious case analysis.

Let  $\alpha\beta\alpha$  be an infix of  $\{\!\{ i \}\!\}_{n+1}$  for some non-empty  $\alpha$  and  $\beta$ . It is enough to show that  $\text{ZType}(\alpha) \leq n + 1$ . By Lemma 16, we only need to consider the case when  $\alpha$  is non-simple. Let  $(\ell, u, r)$  be the parse of  $\alpha$ .

Let  $m$  be an occurrence of  $\alpha\beta\alpha$  in  $\{\!\{ i \}\!\}_{n+1}$ . In particular,  $m$  and  $m + |\alpha\beta|$  are two occurrences of  $\alpha$  in  $\{\!\{ i \}\!\}_{n+1}$ . Hence there are two corresponding occurrences  $m_1$  and  $m_2$  of the parse  $p$  in  $\{\!\{ i \}\!\}_{n+1}$ . Consider the contexts  $c_1$  and  $c_2$  of  $p$  that correspond to the occurrences  $m_1$  and  $m_2$ , respectively. Note that without further hypothesis  $c_1$  and  $c_2$  are not necessarily equal.

We distinguish cases depending on the number of occurrences of a symbol of order  $n + 1$  in  $c_1$ . The cases where  $c_1$  contains 0 or more than 2 symbols of order  $n + 1$  are treated in a similar fashion as in the proof of Theorem 10. Assume that  $c_1$  contains one and only one symbol of order  $n + 1$ .

As  $c_1$  is an infix of  $\{\!\{ i \}\!\}_{n+1}$  with one symbol of order  $n + 1$ , there exists  $k_0 \in [0, \tau(n) - 2]$  and some  $b \in \{0_{n+1}, 1_{n+1}\}$  such that  $c_1 = xby$  where  $x \in \Sigma_n^*$  is a suffix of  $\{\!\{ k_0 \}\!\}_n$  and  $y \in \Sigma_n^*$  is a prefix of  $\{\!\{ k_0 + 1 \}\!\}_n$ .

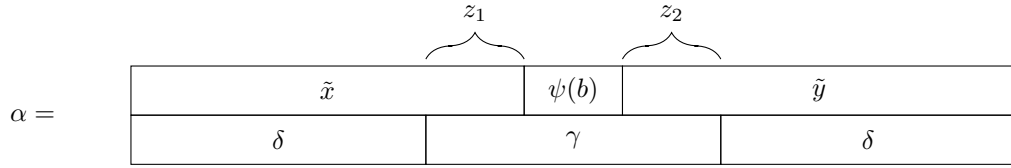
Note that if  $x$  or  $y$  are empty, we can conclude using the induction hypothesis. From now on, we assume that  $x$  and  $y$  are non-empty. In particular, the center  $u$  of the parse  $p = (\ell, u, r)$  contains  $b$  and can therefore be uniquely written as  $u = \underline{x}b\underline{y}$ . Thus,  $c_1 = xby$ ,  $\alpha = \ell\psi(\underline{x})\psi(b)\psi(\underline{y})r$ ,  $x = s\underline{x}$  and  $y = \underline{y}t$  for some  $s$  and  $t$  such that  $s = \varepsilon$  if  $\ell = \varepsilon$  and  $s \in \Sigma$  otherwise, where  $\ell$  is a suffix of  $\psi(s)$  and  $t = \varepsilon$  if  $r = \varepsilon$  and  $t \in \Sigma$  otherwise, where  $r$  is a prefix of  $\psi(t)$ .

**Claim 1.** The context  $c_2$  (of the second occurrence  $m_2$  of  $\alpha$ ) is equal to  $c_1$ .

As  $c_1 = c_2 = xby$  and as  $b$  belongs to the center  $u$  of the parse, the infix  $\alpha$  can be written as  $\alpha = \tilde{x}\psi(b)\tilde{y}$  where  $\tilde{x}$  is a suffix of  $\psi(x)$  and  $\tilde{y}$  is a prefix of  $\psi(y)$ .

**Claim 2.** There exists  $j_0 \in [0, \tau(n) - 1]$  and a non-empty  $\chi$  such that  $\tilde{y}\chi\tilde{x} = \{\!\{ j_0 \}\!\}_n$ .

Let us now consider an arbitrary decomposition of  $\alpha$  as  $\delta\gamma\delta$  for non-empty  $\delta$  and  $\gamma$ . Recall that it is enough to show that  $\text{ZType}(\alpha) \leq n + 1$  or that  $\text{ZType}(\delta) \leq n$ . There are several cases to consider depending on how the decompositions of  $\alpha$  as  $\tilde{x}\psi(b)\tilde{y}$  and  $\delta\gamma\delta$  overlap. We only present here one of the 6 cases where  $|\delta| < |\tilde{x}| \leq |\delta\gamma|$  and  $|\tilde{x}\psi(b)| \leq |\delta\gamma|$ .



In this case  $\gamma = z_1\psi(b)z_2$  with  $z_1 \neq \varepsilon$  such that  $\tilde{x} = \delta z_1$  and  $\tilde{y} = z_2\delta$ . Recall that there exists  $j_0 \in [0, \tau(n) - 1]$  and a non-empty  $\chi$  such that  $\tilde{y}\chi\tilde{x} = \{\{j_0\}_n\}$ . We have  $\text{Zimin}(\{\{j_0\}_n\}) \leq \text{Zimin}(\{\{j_0\}_n\psi(0_{n+1})\}) \leq n + 1$ , where the last inequality follows from the induction hypothesis. Hence,  $\{\{j_0\}_n\} = \tilde{y}\chi\tilde{x} = z_2\delta\chi\delta z_1$  has Zimin index at most  $n + 1$ . This implies that  $\delta$  has Zimin type of at most  $n$  which concludes this case. ◀

**5 Avoiding Zimin patterns in the abelian sense**

Matching a pattern in the abelian sense is a weaker condition, where one only requires that all infixes that are matching a pattern variable must have the same number of occurrences of each letter (instead of being the same words). Hence, for two words  $x, y \in A^*$  we write  $x \equiv y$  if  $|x|_a = |y|_a$  for all  $a \in A$ . Let  $\rho = \rho_1 \cdots \rho_n$  be a pattern, where  $\rho_i \in \mathcal{X}$  is a pattern variable for all  $i \in [n]$ . An *abelian factorization of a word  $w \in A^*$  for the pattern  $\rho$*  is a factorization  $w = w_1 \cdots w_n$  such that  $w_i \neq \varepsilon$  for all  $i \in [n]$  and  $\rho_i = \rho_j$  implies  $w_i \equiv w_j$  for all  $i, j \in [n]$ . A word  $w \in A^*$  *matches the pattern  $\rho$  in the abelian sense* if there is an abelian factorization of  $w$  for  $\rho$ . The definitions when a word encounters a pattern in the abelian sense and when a pattern is unavoidable in the abelian sense are as expected.

We note that every pattern that is unavoidable is in particular unavoidable in the abelian sense. However, the converse does not hold in general as witnessed by the pattern  $xyzxyxuxyzyx$  as shown in [5].

To the best of the authors' knowledge abelian unavoidability still lacks a characterization in the style of general unavoidability in terms of Zimin patterns; we refer to [4] for some open problems and conjectures. Although being possibly less meaningful as for general unavoidability, the analogous Ramsey-like function for abelian unavoidability has been studied. For  $n, k \geq 1$  we define  $g(n, k) = \min\{\ell \geq 1 \mid \forall w \in [k]^\ell : w \text{ encounters } Z_n \text{ in the abelian sense}\}$ . Clearly,  $g(n, k) \leq f(n, k)$  and to the best of the authors' knowledge no elementary upper bound has been shown for  $g$  so far. By applying a combination of the probabilistic method [1] and of analytic combinatorics [7] Tao showed a lower bound for  $g$  given by the first inequality below. Unfortunately, it was not clear to us what the asymptotic behavior of this lower bound is. However Jugé [9] provided us with an estimate of its asymptotic behavior.

► **Theorem 20** (Tao [18], Corollary 3. Jugé [9]). *Let  $k \geq 4$ . Then*

$$g(n, k) \geq (1 + o(1)) \sqrt{2 \prod_{j=1}^{n-1} \left[ \sum_{\ell=1}^{\infty} \frac{1}{k^{2^j \ell}} \sum_{i_1 + \dots + i_k = \ell} \binom{\ell}{i_1, \dots, i_k} \right]^{-1}} \geq \left( \frac{1}{\sqrt{21}} + o(1) \right) \frac{k^{2^{n-1}}}{k^{(n+1)/2}}.$$

In Section 5.1 we prove another doubly-exponential lower bound on  $g$  by applying the first moment method [1]. Our lower bound on  $g$  is not as good as the one obtained by Theorem 20 but its proof seems more direct. The proof follows a similar strategy as the (slightly better) doubly-exponential lower bound for  $f$  from [2], but again, seems to be more direct. Our novel contribution is to provide a matching doubly-exponential upper bound on  $g$  in Section 5.2. Note that Tao in [18] only provides a non-elementary upper bound for the non-abelian case.



### 5.1 A simple lower bound via the first-moment method

For all  $n \geq 1$  let  $\mathcal{X}_n = \{x_1, \dots, x_n\}$  denote the set of the first  $n$  pattern variables. Note that the variable  $x_i$  appears precisely  $2^{n-i}$  times in  $Z_n$  and its first occurrence is at position  $2^{i-1}$  for all  $i \in [1, n]$ . An *abelian occurrence* of  $Z_n$  in a word  $w$  is a pair  $(j, \lambda) \in [0, |w|-1] \times \mathbb{N}^{\mathcal{X}_n}$  for which there is an factorization  $w = uvz$  with  $|u| = j$  and an abelian factorization  $v_1 \cdots v_{2^n-1}$  of  $v$  for  $Z_n$  satisfying  $\lambda(x_i) = |v_{2^{i-1}}|$ .

By applying the probabilistic method [1] we show a lower bound for  $g(n, k)$  that is doubly-exponential in  $n$  for every fixed  $k \geq 2$ . The proof is similar the lower bound proof from [2].

► **Theorem 21.** *For all  $n \geq 1$  and all  $k \geq 2$ ,  $g(n, k) > k^{\lfloor \frac{2^n}{n+2} \rfloor - 1}$ .*

**Proof.** For  $n, \ell \geq 1$  let  $\Delta_{n,k,\ell}$  denote the expected number of abelian occurrences of  $Z_n$  in a random word in the set  $[k]^\ell$ . Note that we always consider the uniform distribution over words. If  $\Delta_{n,k,\ell} < 1$ , then by the probabilistic method [1] there exists a word of length  $\ell$  over the alphabet  $[k]$  that does *not* encounter  $Z_n$  in the abelian sense; hence we can conclude that  $g(n, k) > \ell$ . Therefore we investigate those  $\ell = \ell(n, k)$  for which we can guarantee that  $\Delta_{n,k,\ell} < 1$ . We need two intermediate claims.

**Claim 1.** Let  $A_{k,h}$  denote the event that two independent random words  $u$  and  $v$  in  $[k]^h$  satisfy  $u \equiv v$ . Then  $\Pr(A_{k,h}) \leq 1/k$  for all  $h \geq 1$ .

It follows that the probability that  $m$  random words  $w_1, w_2, \dots, w_m \in [k]^h$  satisfy  $w_1 \equiv w_2 \equiv \dots \equiv w_m$  is at most  $(1/k)^{m-1}$ . Recall that  $Z_n = y_1 \cdots y_{2^n-1}$ , where  $y_i \in \{x_1, \dots, x_n\}$  for all  $i \in [2^n - 1]$  and that the variable  $x_i$  appears precisely  $2^{n-i}$  times in  $Z_n$ . We recall that we would like to bound the expected number of occurrences (in the abelian sense) of  $Z_n$  in a random word of length  $\ell$  over the alphabet  $[k]$ . To account for this, we define for each mapping  $\lambda : \mathcal{X}_n \rightarrow \mathbb{N}^+$  its *width* as  $\text{width}(\lambda) = \sum_{i=1}^n 2^{n-i} \cdot \lambda(x_i)$ . For every word  $v$  of length  $\text{width}(\lambda)$  its (unique) *decomposition with respect to  $\lambda$*  is the unique factorization  $v = v_1 \cdots v_{2^n-1}$  such that  $y_j = x_i$  implies  $|v_j| = \lambda(x_i)$  for all  $j \in [2^n - 1]$  and all  $i \in [n]$ . Using the estimations in Claim 1 one can now show the following claim.

**Claim 2.** Let  $\lambda : \mathcal{X}_n \rightarrow \mathbb{N}^+$  of width  $d$  and let  $B_\lambda$  denote the event that  $(0, \lambda)$  is an occurrence in the abelian sense of  $Z_n$  in a random word from  $[k]^d$ . Then  $\Pr(B_\lambda) \leq k^{n-2^n+1}$ .

The probability that  $(j, \lambda)$  is an occurrence of  $Z_n$  in a random word from  $[k]^\ell$  with  $\ell \geq j + \text{width}(\lambda)$  is equal to the probability that  $(0, \lambda)$  is an occurrence of  $Z_n$  in a random word from  $[k]^d$  (which is  $\Pr(B_\lambda)$ ). Thus, this probability does not depend on  $j$ .

We are ready to prove an upper bound for  $\Delta_{n,k,\ell}$ , keeping in mind that any occurrence  $(j, \lambda)$  of  $Z_n$  in a random word of length  $\ell$  must satisfy  $\text{width}(\lambda) \geq 2^n - 1$ .

$$\begin{aligned} \Delta_{n,k,\ell} &\leq \sum_{d=2^n-1}^{\ell} \sum_{j=0}^{\ell-d} \sum_{\substack{\lambda: \mathcal{X}_n \rightarrow \mathbb{N}^+ \\ \text{width}(\lambda)=d}} \Pr[(j, \lambda) \text{ is an ab. occ. of } Z_n \text{ in a random word in } [k]^\ell] \\ &\leq \sum_{d=2^n-1}^{\ell} \sum_{j=0}^{\ell-d} \sum_{\substack{\lambda: \mathcal{X}_n \rightarrow \mathbb{N}^+ \\ \text{width}(\lambda)=d}} \Pr(B_\lambda) \stackrel{\text{Claim 2}}{\leq} \sum_{d=2^n-1}^{\ell} \sum_{j=0}^{\ell-d} \sum_{\substack{\lambda: \mathcal{X}_n \rightarrow \mathbb{N}^+ \\ \text{width}(\lambda)=d}} k^{n-2^n+1} \\ &\leq \sum_{d=2^n-1}^{\ell} \sum_{j=0}^{\ell-d} d^n \cdot k^{n-2^n+1} \leq \frac{\ell^2 \cdot \ell^n}{k^{2^n-n-1}} = \frac{\ell^{n+2}}{k^{2^n-n-1}} \end{aligned} \tag{1}$$

It remains to determine a sufficiently large  $\ell$  that guarantees  $\Delta_{n,k,\ell} < 1$ . Using the previous inequalities it is not difficult to show that  $\ell = k^{\lfloor \frac{2^n}{n+2} \rfloor - 1}$  ensures that  $\Delta_{n,k,\ell} < 1$ . ◀

## 5.2 A doubly-exponential upper bound

Let us finally prove an upper bound for  $g(n, k)$  that is doubly-exponential in  $n$ .

► **Theorem 22.** *For all  $n, k \geq 1$ ,  $g(n, k) \leq 2^{(4k)^n (n-1)!}$ .*

**Proof.** For all  $n \geq 1$ , we will show the following inequality:

$$g(n+1, k) \leq (g(n, k) + 1)(g(n, k)^{kn} + 1) \tag{2}$$

A simple induction on  $n$  then shows the claimed upper bound.

To show (2), we consider words  $w$  over  $[k]$  of length at least  $(g(n, k) + 1)(g(n, k)^{kn} + 1)$ . Such words can always be written as  $w_1 a_1 w_2 a_2 \cdots w_m a_m z$ , where  $m = g(n, k)^{kn} + 1$ ,  $|w_j| = g(n, k)$  and  $a_j \in [k]$  for all  $j \in [m]$  and  $z \in [k]^*$ .

By definition of  $g(n, k)$ , for all  $j \in [m]$ , the word  $w_j$  encounters  $Z_n$  in the abelian sense, witnessed in some infix  $v_j$  by some abelian factorization  $v_j = v_j^{(1)} \cdots v_j^{(2^n - 1)}$  for  $Z_n$ . For each such abelian factorization, it is natural to associate with every  $i \in [n]$ , the (unique) Parikh image of the words  $v_j^{(h)}$  of the factorization that correspond to the different occurrences of the variable  $x_i$  in  $Z_n$ . Formally, each of the above abelian factorizations  $v_j = v_j^{(1)} \cdots v_j^{(2^n - 1)}$  induces a mapping  $\psi_j : \mathcal{X}_n \rightarrow \mathbb{N}^{[k]}$  such that  $\psi_j(x_i)(a) = |v_j^{(2^i - 1)}|_a$  for all  $j \in [m]$ , all  $i \in [n]$  and all  $a \in [k]$ . As expected, we write  $\psi_j \equiv \psi_h$  if  $\psi_j(x_i) = \psi_h(x_i)$  for all  $i \in [n]$ . Note that if there are distinct  $j, h \in [1, m]$  with  $\psi_j \equiv \psi_h$ , then  $w$  encounters  $Z_{n+1} = Z_n x_{n+1} Z_n$  in the abelian sense. It is easy to see that there are at most  $g(n, k)^{kn}$  different equivalence classes for the  $\psi_j$  with respect to  $\equiv$ . Therefore as  $m = g(n, k)^{kn} + 1$ , there are always two distinct indices  $i, j \in [1, m]$  that satisfy  $\psi_i \equiv \psi_j$  and we have established (2). ◀

## 6 Conclusion

We have established a lower bound for  $f(n, k)$  that is already non-elementary when  $k = 2$ . A natural question is whether the non-elementary lower bound for  $f(n, k)$  obtained by an explicit construction in this article can be obtained using the probabilistic method. A first hint of an answer is that the first moment method used in [2] cannot be used to obtain a lower bound that is asymptotically above doubly-exponential. Indeed, as for a length  $\ell \geq k^{2^n - n - 2} + 2^n$ , the expected number  $\Delta_{n,k,\ell}$  of occurrences  $Z_n$  in a random word in  $[k]^\ell$  is greater than 1.

To see this, recall that  $|Z_n| = 2^n - 1$  and hence there is at most one possible occurrence of  $Z_n$  in any word of length  $2^n - 1$ . Let  $A_n$  denote the event that  $Z_n$  is encountered in a random word in  $[k]^{2^n - 1}$ . We have  $\Pr(A_n) = \prod_{i=1}^n (1/k)^{2^{n-i} - 1} = k^{-2^n + n + 2}$ .

Assume that  $\ell \geq k^{2^n - n - 2} + 2^n$ . For each  $i \in [0, k^{2^n - n - 2}]$ , let  $X_i$  be the indicator random variable marking that the infix, of a random word in  $[k]^\ell$ , occurring at  $i$  and of length  $2^n - 1$  matches  $Z_n$ . By linearity of the expectation, the following lower bound holds,  $\Delta_{n,k,\ell} \geq \sum_{i=0}^K E(X_i) \geq (K + 1) \Pr(A_n) = 1 + \frac{1}{K} \geq 1$ , where  $K = k^{2^n - n - 2}$ .

**Acknowledgments.** The authors would like to thank Vincent Jugé for his help and the anonymous reviewers for their valuable feedback.

---

**References**

---

- 1 N. Alon and J. Spencer. *The Probabilistic Method*. Wiley, 2015.
- 2 J. Cooper and D. Rorabaugh. Bounds on Zimin word avoidance. *CoRR*, abs/1409.3080, 2014. URL: <http://arxiv.org/abs/1409.3080>.
- 3 J. Cooper and D. Rorabaugh. Asymptotic density of Zimin words. *Discrete Mathematics & Theoretical Computer Science*, Vol. 18, no 3, 2016. URL: <http://dmtcs.episciences.org/1414>.
- 4 J. D. Currie. Pattern avoidance: themes and variations. *Theor. Comput. Sci.*, 339(1):7–18, 2005. doi:10.1016/j.tcs.2005.01.004.
- 5 J. D. Currie and V. Linek. Avoiding patterns in the abelian sense. *Canadian J. Math.*, 51(4):696–714, 2001. doi:10.4153/cjm-2001-028-4.
- 6 G. F. McNulty D. R. Bean, A. Ehrenfeucht. Avoidable Patterns in Strings of Symbols. *Pac. J. of Math.*, 85:261–294, 1979. doi:10.2140/pjm.1979.85.261.
- 7 P. Flajolet and R. Sedgewick. *Analytic combinatorics*. Cambridge University Press, Cambridge, 2009.
- 8 P. Jancar. Equivalences of pushdown systems are hard. In *Proceedings of FOSSACS 2014*, volume 8412 of *Lecture Notes in Computer Science*, pages 1–28. Springer, 2014. doi:10.1007/978-3-642-54830-7\_1.
- 9 V. Jugé. Abelian Ramsey length and asymptotic lower bounds. *CoRR*, abs/1609.06057, 2016. URL: <http://arxiv.org/abs/1609.06057>.
- 10 K. Reinhardt. The complexity of translating logic to finite automata. In *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*, pages 231–238. Springer, 2001. doi:10.1007/3-540-36387-4\_13.
- 11 D. Rorabaugh. Toward the combinatorial limit theory of free words. *CoRR*, abs/1509.04372, 2015. URL: <http://arxiv.org/abs/1509.04372>.
- 12 W. Rytter and A. M. Shur. Searching for Zimin patterns. *Theor. Comput. Sci.*, 571:50–57, 2015. doi:10.1016/j.tcs.2015.01.004.
- 13 M. V. Sapir. Combinatorics on words with applications. Technical report, LITP, 1995.
- 14 S. Schmitz. Complexity hierarchies beyond elementary. *CoRR*, abs/1312.5686, 2014. URL: <http://arxiv.org/abs/1312.5686>.
- 15 G. Sénizergues. The equivalence problem for t-turn DPDA is co-NP. Technical Report 1297-03, LaBRI, 2003. available at <http://dept-info.labri.u-bordeaux.fr/~ges>.
- 16 C. Stirling. Deciding DPDA equivalence is primitive recursive. In *In Proceedings of ICALP 2002*, volume 2380 of *Lecture Notes in Computer Science*, pages 821–832. Springer, 2002. doi:10.1007/3-540-45465-9\_70.
- 17 L. J. Stockmeyer. *The complexity of decision problems in automata and logic*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1974.
- 18 J. Tao. Pattern occurrence statistics and applications to the ramsey theory of unavoidable patterns. *CoRR*, abs/1406.0450, 2014. URL: <http://arxiv.org/abs/1406.0450>.
- 19 A. Thue. Über unendliche Zeichenreihen. *Norske Vid. Skrifter I Mat.-Nat. Kl. Christiania*, 7:1–22, 1906.
- 20 A. I. Zimin. Blocking sets of terms. *Math. USSR Sbornik*, 47:50–57, 1984. doi:10.1070/sm1984v047n02abeh002647.



# Extended Learning Graphs for Triangle Finding\*

Titouan Carette<sup>1</sup>, Mathieu Laurière<sup>2</sup>, and Frédéric Magniez<sup>3</sup>

1 Ecole Normale Supérieure de Lyon, Lyon, France

titouan.carette@ens-lyon.fr

2 NYU-ECNU Institute of Mathematical Sciences at NYU Shanghai, Shanghai, China

mathieu.lauriere@nyu.edu

3 CNRS, IRIF, Univ Paris Diderot, Paris, France

frederic.magniez@cnrs.fr

---

## Abstract

We present new quantum algorithms for Triangle Finding improving its best previously known quantum query complexities for both dense and sparse instances. For dense graphs on  $n$  vertices, we get a query complexity of  $O(n^{5/4})$  without any of the extra logarithmic factors present in the previous algorithm of Le Gall [FOCS'14]. For sparse graphs with  $m \geq n^{5/4}$  edges, we get a query complexity of  $O(n^{11/12}m^{1/6}\sqrt{\log n})$ , which is better than the one obtained by Le Gall and Nakajima [ISAAC'15] when  $m \geq n^{3/2}$ . We also obtain an algorithm with query complexity  $O(n^{5/6}(m \log n)^{1/6} + d_2\sqrt{n})$  where  $d_2$  is the variance of the degree distribution.

Our algorithms are designed and analyzed in a new model of learning graphs that we call extended learning graphs. In addition, we present a framework in order to easily combine and analyze them. As a consequence we get much simpler algorithms and analyses than previous algorithms of Le Gall *et al* based on the MNRS quantum walk framework [SICOMP'11].

**1998 ACM Subject Classification** F.2 Analysis of Algorithms and Problem Complexity, F.1.1 Models of Computation

**Keywords and phrases** Quantum query complexity, learning graphs, triangle finding

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.20

## 1 Introduction

Decision trees form a simple model for computing Boolean functions by successively reading the input bits until the value of the function can be determined. In this model, the *query complexity* is the number of input bits queried. This allows us to study the complexity of a function in terms of its structural properties. For instance, sorting an array of size  $n$  can be done using  $O(n \log n)$  comparisons, and this is optimal for comparison-only algorithms.

In an extension of the deterministic model, one can also allow randomized and even quantum computations. Then the speed-up can be exponential for partial functions (*i.e.* problems with promise) when we compare deterministic with randomized computation, and randomized with quantum computation. The case of total functions is rather fascinating. For them, the best possible gap can only be polynomial between each models [20, 4], which is still useful in practice for many problems. But surprisingly, the best possible gap is still an open question, even if it was improved for both models very recently [3, 1]. In the context of quantum computing, query complexity captures the great algorithmic successes of quantum

---

\* This work has been partially supported by the European Commission project Quantum Algorithms (QALGO) and the French ANR Blanc project RDAM.



computing like the search algorithm of Grover [13] and the period finding subroutine of Shor’s factoring algorithm [22], while at the same time it is simple enough that one can often show tight lower bounds.

Reichardt [21] showed that the general adversary bound, formerly just a lower bound technique for quantum query complexity [14], is also an upper bound. This characterization has opened an avenue for designing quantum query algorithms. However, even for simple functions it is challenging to find an optimal bound. Historically, studying the query complexity of specific functions led to amazing progresses in our understanding of quantum computation, by providing new algorithmic concepts and tools for analyzing them. Some of the most famous problems in that quest are Element Distinctness and Triangle Finding [10]. Element Distinctness consists in deciding if a function takes twice the same value on a domain of size  $n$ , whereas Triangle Finding consists in determining if an  $n$ -vertex graph has a triangle. Quantum walks were used to design algorithms with optimal query complexity for Element Distinctness. Later on, a general framework for designing quantum walk based algorithms was developed with various applications [18], including for Triangle Finding [19].

For seven years, no progress on Triangle Finding was done until Belovs developed his beautiful model of *learning graphs* [5], which can be viewed as the minimization form of the general adversary bound with an additional structure imposed on the form of the solution. This structure makes learning graphs easier to reason about without any background on quantum computing. On the other hand, they may not provide always optimal algorithms. Learning graphs have an intuitive interpretation in terms of electrical networks [7]. Their complexity is related to the total conductance of the underlying network and its effective resistance. Moreover this characterization leads to a generic quantum implementation based on a quantum version of random walks which is time efficient and preserves query complexity.

Among other applications, learning graphs have been used to design an algorithm for Triangle Finding with query complexity  $O(n^{35/27})$  [6], improving on the previously known bound  $\tilde{O}(n^{1.3})$  obtained by a quantum walk based algorithm [19]. Then the former was improved by another learning graph using  $O(n^{9/7})$  queries [16]. This learning graph has been proven optimal for the original class of learning graphs [9], known as *non-adaptive learning graphs*, for which the conductance of each edge is constant. Then, Le Gall showed that quantum walk based algorithms are indeed stronger than non-adaptive learning graphs for Triangle Finding by constructing a new quantum algorithm with query complexity  $\tilde{O}(n^{5/4})$  [11]. His algorithm combines in a novel way combinatorial arguments on graphs with quantum walks. One of the key ingredient is the use of an algorithm due to Ambainis for implementing Grover Search in a model whose queries may have variable complexities [2]. Le Gall used this algorithm to average the complexity of different branches of its quantum walk in a quite involved way. In the specific case of sparse graphs, those ideas have also demonstrated their advantage for Triangle Finding on previously known algorithms [12].

The starting point of the present work is to investigate a deeper understanding of learning graphs and their extensions. Indeed, various variants have been considered without any unified and intuitive framework. For instance, the best known quantum algorithm for  $k$ -Element Distinctness (a variant of Element Distinctness where we are now checking if the function takes  $k$  times the same value) has been designed by several clever relaxations of the model of learning graphs [5]. Those relaxations led to algorithms more powerful than non-adaptive learning graphs, but at the price of a more complex and less intuitive analysis. In **Section 3**, we extract several of those concepts that we formalize in our new model of *extended learning graphs* (**Definition 3.1**). We prove that their complexity (**Definition 3.2**) is always an upper bound on the query complexity of the best quantum algorithm solving

the same problem (**Theorem 3.3**). We also introduce the useful notion of *super edge* (**Definition 3.4**) for compressing some given portion of a learning graph. We use them to encode efficient learning graphs querying a part of the input on some given index set (**Lemmas 3.7 and 3.8**). In some sense, we transpose to the learning graph setting the strategy of finding all 1-bits of some given sparse input using Grover Search.

In **Section 4**, we provide several tools for composing our learning graphs. We should first remind the reader that, since extended learning graphs cover a restricted class of quantum algorithms, it is not possible to translate all quantum algorithms in that model. Nonetheless we succeed for two important algorithmic techniques: Grover Search with variable query complexities [2] (**Lemma 4.1**), and Johnson Walk based quantum algorithms [19, 18] (**Theorem 4.2**). In the last case, we show how to incorporate the use of super edges for querying sparse inputs.

We validate the power and the ease of use of our framework on Triangle Finding in **Section 5**. First, denoting the number of vertices by  $n$ , we provide a simple adaptive learning graph with query complexity  $O(n^{5/4})$ , whose analysis is arguably much simpler than the algorithm of Le Gall, and whose complexity is cleared of logarithmic factors (**Theorem 5.1**). This also provides a natural separation between non-adaptive and adaptive learning graphs. Then, we focus on sparse input graphs and develop extended learning graphs. All algorithms of [9] could be rephrased in our model. But more importantly, we show that one can design more efficient ones. For sparse graphs with  $m \geq n^{5/4}$  edges, we get a learning graph with query complexity  $O(n^{11/12}m^{1/6}\sqrt{\log n})$ , which improves the results of [12] when  $m \geq n^{3/2}$  (**Theorem 5.2**). We also construct another learning graph with query complexity  $O(n^{5/6}(m \log n)^{1/6} + d_2\sqrt{n})$ , where  $d_2$  is the variance of the degree distribution (**Theorem 5.3**). To the best of our knowledge, this is the first quantum algorithm for Triangle Finding whose complexity depends on this parameter  $d_2$ .

## 2 Preliminaries

We will deal with Boolean functions of the form  $f : Z \rightarrow \{0, 1\}$ , where  $Z \subseteq \{0, 1\}^N$ . In the query model, given a function  $f : Z \rightarrow \{0, 1\}$ , the goal is to evaluate  $f(z)$  by making as few queries to the  $z$  as possible. A query is a question of the form ‘What is the value of  $z$  in position  $i \in [N]$ ?’ to which is returned  $z_i \in \{0, 1\}$ .

In this paper we will discuss functions whose inputs are graphs viewed through their adjacency matrices. Then  $z$  will encode an undirected graph  $G$  on vertex set  $[n]$ , that is  $N = \binom{n}{2}$  in order to encode the possible edges of  $G$ . Then  $z_{ij} = 1$  iff  $ij$  is an edge of  $G$ .

In the quantum query model, these queries can be asked in superposition. We refer the reader to the survey [15] for precise definitions and background on the quantum query model. We denote by  $Q(f)$  the number of queries needed by a quantum algorithm to evaluate  $f$  with error at most  $1/3$ . Surprisingly, the general adversary bound, that we define below, is a tight characterization of  $Q(f)$ .

► **Definition 2.1.** Let  $f : Z \rightarrow \{0, 1\}$  be a function, with  $Z \subseteq \{0, 1\}^N$ . The *general adversary bound*  $\text{Adv}^\pm(f)$  is defined as the optimal value of the following optimization problem:

$$\text{minimize: } \max_{z \in Z} \sum_{j \in [N]} X_j[z, z] \quad \text{subject to: } \sum_{j \in [N]: x_j \neq y_j} X_j[x, y] = 1, \text{ when } f(x) \neq f(y), \\ X_j \succeq 0, \forall j \in [N],$$

where the optimization is over positive semi-definite matrices  $X_j$  with rows and columns labeled by the elements of  $Z$ , and  $X_j[x, y]$  is used to denote the  $(x, y)$ -entry of  $X_j$ .

► **Theorem 2.2** ([14, 17, 21]).  $Q(f) = \Theta(\text{Adv}^\pm(f))$ .

### 3 Extended learning graphs

Consider a Boolean function  $f : Z \rightarrow \{0, 1\}$ , where  $Z \subseteq \{0, 1\}^N$ . The set of positive inputs (or instances) will be usually denoted by  $Y = f^{-1}(1)$ . A *1-certificate* for  $f$  on  $y \in Y$  is a subset  $I \subseteq [N]$  of indices such that  $f(z) = 1$  for every  $z \in Z$  with  $z_I = y_I$ , where  $z_I = (z_i)_{i \in I}$ .

#### 3.1 Model and complexity

Intuitively, learning graphs are simply electric networks of a special type. The network is embedded in a rooted directed acyclic graph, which has few similarities with decision trees. Vertices are labelled by subsets  $S \subseteq [n]$  of indices. Edges are basically from any vertex labelled by, say,  $S$  to any other one labelled  $S \cup \{j\}$ , for some  $j \notin S$ . Such an edge can be interpreted as querying the input bit  $x_j$ , while  $x_S$  has been previously learnt. The weight on the edge is its conductance: the larger it is, the more flow will go through it. Sinks of the graph are labelled by potential 1-certificates of the function we wish to compute. Thus a random walk on that network starting from the root (labelled by  $\emptyset$ ), with probability transitions proportional to conductances, will hit a 1-certificate with average time proportional to the product of the total conductance by the effective resistance between the root of leaves having 1-certificates [7]. If weights are independent of the input, then the learning graph is called *non-adaptive*. When they depend on previously learned bits, it is *adaptive*. But in quantum computing, we will see that they can also depend on both the value of the next queried bit and the value of the function itself! We call them *extended learning graphs*.

Formally, we generalize the original model of learning graphs by allowing two possible weights on each edge: one for positive instances and one for negative ones. Those weights are linked together as explained in the following definition.

► **Definition 3.1** (Extended learning graph). Let  $Y \subseteq Z$  be finite sets. An *extended learning graph*  $\mathcal{G}$  is a 5-tuple  $(\mathcal{V}, \mathcal{E}, \mathcal{S}, \{w_z^b : z \in Z, b \in \{0, 1\}\}, \{p_y : y \in Y\})$  satisfying

- $(\mathcal{V}, \mathcal{E})$  is a directed acyclic graph rooted in some vertex  $r \in \mathcal{V}$ ;
- $\mathcal{S}$  is a vertex labelling mapping each  $v \in \mathcal{V}$  to  $\mathcal{S}(v) \subseteq [N]$  such that  $\mathcal{S}(r) = \emptyset$  and  $\mathcal{S}(v) = \mathcal{S}(u) \cup \{j\}$  for every  $(u, v) \in \mathcal{E}$  and some  $j \notin \mathcal{S}(u)$ ;
- Values  $w_z^b(u, v)$  are in  $\mathbb{R}_{\geq 0}$  and depend on  $z$  only through  $z_{\mathcal{S}(v)}$ , for every  $(u, v) \in \mathcal{E}$ ;
- $w_x^0(u, v) = w_y^1(u, v)$  for all  $x \in Z \setminus Y, y \in Y$  and edges  $(u, v) \in \mathcal{E}$  such that  $x_{\mathcal{S}(u)} = y_{\mathcal{S}(u)}$  and  $x_j \neq y_j$  with  $\mathcal{S}(v) = \mathcal{S}(u) \cup \{j\}$ .
- $p_y : \mathcal{E} \rightarrow \mathbb{R}_{\geq 0}$  is a unit flow whose source is the root and such that  $p_y(e) = 0$  when  $w_y^1(e) = 0$ , for every  $y \in Y$ .

We say that  $\mathcal{G}$  is a *learning graph* for some function  $f : Z \rightarrow \{0, 1\}$ , when  $Y = f^{-1}(1)$  and each sink of  $p_y$  contains a 1-certificate for  $f$  on  $y$ , for all positive input  $y \in f^{-1}(1)$ .

We also say that  $\mathcal{G}$  is an *adaptive learning graph* when  $w_z^0 = w_z^1$  for all  $z \in Z$ . If furthermore  $w_z^0$  is independent of  $z$ ,  $\mathcal{G}$  is a *non-adaptive learning graph*. Unless otherwise specified, by *learning graph* we mean *extended learning graph*.

When there is no ambiguity, we usually define  $\mathcal{S}$  by stating the *label* of each vertex. We also say that an edge  $e = (u, v)$  *loads*  $j$  when  $\mathcal{S}(v) = \mathcal{S}(u) \cup \{j\}$ . A *transition of length*  $k$  is a sequence of edges of the form  $((v_1, v_2), (v_2, v_3), \dots, (v_{k-1}, v_k))$ , with  $v_i \neq v_j$  for  $i \neq j$ .

The complexity of extended learning graphs is defined similarly to the one of other learning graphs by choosing the appropriate weight function for each complexity terms.

► **Definition 3.2** (Extended learning graph complexity). Let  $\mathcal{G}$  be an extended learning graph for a function  $f : Z \rightarrow \{0, 1\}$ . Let  $x \in Z \setminus f^{-1}(1)$ ,  $y \in f^{-1}(1)$ , and  $\mathcal{F} \subseteq \mathcal{E}$ . The *negative*



complexity of  $\mathcal{F}$  on  $x$  and the *positive complexity* of  $\mathcal{F}$  on  $y$  (with respect to  $\mathcal{G}$ ) are respectively defined by

$$C^0(\mathcal{F}, x) = \sum_{e \in \mathcal{F}} w_x^0(e) \quad \text{and} \quad C^1(\mathcal{F}, y) = \sum_{e \in \mathcal{F}} \frac{p_y(e)^2}{w_y^1(e)}.$$

Then the *negative and positive complexities* of  $\mathcal{F}$  are  $C^0(\mathcal{F}) = \max_{x \in f^{-1}(0)} C^0(\mathcal{F}, x)$  and  $C^1(\mathcal{F}) = \max_{y \in f^{-1}(1)} C^1(\mathcal{F}, y)$ . The *complexity* of  $\mathcal{F}$  is  $C(\mathcal{F}) = \sqrt{C^0(\mathcal{F})C^1(\mathcal{F})}$  and the *complexity* of  $\mathcal{G}$  is  $C(\mathcal{G}) = C(\mathcal{E})$ . Last, the *extended learning graph complexity* of  $f$ , denoted  $\mathcal{LG}^{\text{ext}}(f)$ , is the minimum complexity of an extended learning graph for  $f$ .

Most often we will split a learning graph into *stages*  $\mathcal{F}$ , that is, when the flow through  $\mathcal{F}$  has the same total amount 1 for every positive input. This allows us to analyze the learning graph separately on each stage.

As for adaptive learning graphs [6, 8], the extended learning graph complexity is upper bounding the standard query complexity.

► **Theorem 3.3.** *For every function  $f : Z \rightarrow \{0, 1\}$ , we have  $Q(f) = O(\mathcal{LG}^{\text{ext}}(f))$ .*

**Proof.** We assume that  $f$  is not constant, otherwise the result holds readily. The proof follows the lines of the analysis of the learning graph for Graph collision in [5]. We already know that  $Q(f) = O(\text{Adv}^\pm(f))$  by Theorem 2.2. Fix any extended learning graph  $\mathcal{G}$  for  $f$ . Observe from Definition 2.1 that  $\text{Adv}^\pm(f)$  is defined by a minimization problem. Therefore finding any feasible solution with objective value  $C(\mathcal{G}, f)$  would conclude the proof. Without loss of generality, assume that  $C^0(\mathcal{G}) = C^1(\mathcal{G})$  (otherwise we can multiply all weights by  $\sqrt{C^1(\mathcal{G})/C^0(\mathcal{G})}$ ). Then both complexities become  $\sqrt{C^0(\mathcal{G})C^1(\mathcal{G})}$  and the total complexity remains  $C(\mathcal{G})$ .

For each edge  $e = (u, v) \in \mathcal{E}$  with  $\mathcal{S}(v) = \mathcal{S}(u) \cup \{j\}$ , define a block-diagonal matrix  $X_j^e = \sum_\alpha (Y_j^e)_\alpha$ , where the sum is over all possible assignments  $\alpha$  on  $\mathcal{S}(u)$ . Each  $(Y_j^e)_\alpha$  is defined as  $(\psi_0 \psi_0^* + \psi_1 \psi_1^*)$ , where for each  $z \in \{0, 1\}^n$  and  $b \in \{0, 1\}$

$$\psi_b[z] = \begin{cases} p_z(e)/\sqrt{w_z^1(e)} & \text{if } z_{\mathcal{S}(u)} = \alpha, f(z) = 1 \text{ and } z_j = 1 - b, \\ \sqrt{w_z^0(e)} & \text{if } z_{\mathcal{S}(u)} = \alpha, f(z) = 0 \text{ and } z_j = b, \\ 0 & \text{otherwise.} \end{cases}$$

Define now  $X_j = \sum_e X_j^e$  where the sum is over all edges  $e$  loading  $j$ . Fix any  $x \in f^{-1}(0)$  and  $y \in f^{-1}(1)$ . Then we have  $X_j^e[x, x] = w_x^0(e)$  and  $X_j^e[y, y] = (p_y(e))^2/w_y^1(e)$ . So the objective value is

$$\begin{aligned} \max_{z \in \{0, 1\}^n} \sum_{j \in [N]} X_j[z, z] &= \max \left\{ \max_{x \in f^{-1}(0)} \sum_j X_j[x, x], \max_{y \in f^{-1}(1)} \sum_j X_j[y, y] \right\} \\ &= \max \{C^0(\mathcal{G}), C^1(\mathcal{G})\} = C(\mathcal{G}). \end{aligned}$$

Consider the cut  $\mathcal{F}$  over  $\mathcal{G}$  of edges  $(u, v) \in \mathcal{E}$  such that  $\mathcal{S}(v) = \mathcal{S}(u) \cup \{j\}$  and  $x_{\mathcal{S}(u)} = y_{\mathcal{S}(u)}$  but  $x_j \neq y_j$ . Then each edge  $e \in \mathcal{F}$  loading  $j$  satisfies  $w_x^0(e) = w_y^1(e)$  and therefore  $X_j^e[x, y] = p_y(e)$ . Thus,  $\sum_{j: x_j \neq y_j} X_j[x, y] = \sum_{e \in \mathcal{F}} p_y(e) = 1$ . Hence the constraints of Definition 2.1 are satisfied. ◀

### 3.2 Compression of learning graphs into super edges

We will simplify the presentation of our learning graphs by introducing a new type of edge encoding specific learning graphs as sub-procedures. Since an edge has a single ‘exit’, we can only encode learning graphs whose flows have unique sinks.

► **Definition 3.4** (Super edge). A *super edge*  $e$  is an extended learning graph  $\mathcal{G}_e$  such that each possible flow has the same unique sink. Then its *positive* and *negative edge-complexities* on input  $x \in Z \setminus Y$  and  $y \in Y$  are respectively  $c^0(e, x) = C^0(\mathcal{G}_e, x)$  and  $c^1(e, y) = C^1(\mathcal{G}_e, y)$ .

Consider an edge  $e$  of a learning graph  $\mathcal{G}$  with flow  $p$ . We can view  $e$  as a super edge with  $c^0(e, x) = w_x^0(e)$  and  $c^1(e, y) = 1/w_y^1(e)$ . We have to take into account the fact that, in  $\mathcal{G}$ , its flow is not 1 but  $p_z(e)$  for each  $z$ , so  $C^0(e, x) = c^0(e, x)$  and  $C^1(e, y) = p_y(e)^2 \times c^1(e, y)$ . We use these notions in order to define the complexity of learning graphs with super edges.

Any learning graph with super edges is equivalent to a learning graph without super edges by doing recursively the following replacement for each super edge  $e = (u_e, v_e)$ : (1) replace it by its underlying learning graph  $\mathcal{G}_e$ , plugging the root to all incoming edges and the unique flow sink to all outgoing edges, and changing the labels as follows: we enrich the labels of vertices in  $\mathcal{G}_e$  using the label of  $u_e$ , that is, if  $\mathcal{S}$  and  $\mathcal{S}_e$  are the vertex labelling mappings of  $\mathcal{G}$  and  $\mathcal{G}_e$  respectively, for each vertex  $w$  of  $\mathcal{G}_e$  the label becomes  $\mathcal{S}_e(w) \cup \mathcal{S}(u)$ ; (2) root the incoming flow as in  $\mathcal{G}_e$ . Let us call this learning graph the *expansion* of the original one with super edges. Then, a direct inspection leads to the following result that we will use in order to compute complexities directly on our (extended) learning graphs.

► **Lemma 3.5.** *Let  $\mathcal{G}$  be a learning graph with super edges for some function  $f$ . Then the expansion of  $\mathcal{G}$  is also a learning graph for  $f$ . Moreover, let  $\text{exp}(\mathcal{F})$  be the expansion of  $\mathcal{F} \subseteq \mathcal{E}$ . Then  $\text{exp}(\mathcal{F})$  has positive and negative complexities*

$$C^0(\text{exp}(\mathcal{F}), x) = \sum_{e \in \mathcal{F}} c^0(e, x) \quad \text{and} \quad C^1(\text{exp}(\mathcal{F}), y) = \sum_{e \in \mathcal{F}} p_y(e)^2 \times c^1(e, y).$$

Fix some stage  $\mathcal{F} \subseteq \mathcal{E}$  of  $\mathcal{G}$  such that the flow through  $\mathcal{F}$  has total amount 1 for each positive input. We will use the following lemma (adapted from non-adaptive learning graphs) to assume that a learning graph has positive complexity at most 1 on  $\mathcal{F}$ . The expectation involved here comes from the factor  $T$ , which is a parameter called the *speciality* of  $\mathcal{F}$ .

► **Lemma 3.6** (Speciality [5]). *Let  $\mathcal{G}$  be a learning graph for  $f : Z \rightarrow \{0, 1\}$ . Let  $\mathcal{F} \subseteq \mathcal{E}$  be a stage of  $\mathcal{G}$  whose flow always uses the ratio  $1/T$  of transitions and every transition receives the same amount of flow. Then there is a learning graph  $\tilde{\mathcal{F}}$  composed of the edges of  $\mathcal{F}$  equipped with new weights such that, denoting  $c^1(e) = \max_{y' \in f^{-1}(1)} c^1(e, y')$ ,*

$$C^0(\tilde{\mathcal{F}}, x) \leq T \mathbb{E}_{e \in \mathcal{F}} [c^0(e, x) c^1(e)] \quad \text{and} \quad C^1(\tilde{\mathcal{F}}, y) \leq 1, \quad \forall x \in f^{-1}(0), y \in f^{-1}(1).$$

**Proof.** Let  $n_{\text{total}}$  be the number of transitions in  $\mathcal{F}$  and  $n_{\text{used}}$  the number of them used by each flow (i.e. with positive flow). Therefore  $T = n_{\text{total}}/n_{\text{used}}$ . By assumption, the flow on each edge is either 0 or  $1/n_{\text{used}}$ . For each edge  $e$  in  $\mathcal{F}$ , let  $\lambda_e = c^1(e)/n_{\text{used}}$ . For every input  $z$ , we multiply  $w_z^b(e)$  by  $\lambda_e$ , and we name  $\tilde{\mathcal{F}}$  the set  $\mathcal{F}$  with the new weights. Then for any  $x \in f^{-1}(0)$ ,  $C^0(\tilde{\mathcal{F}}, x) = \sum_{e \in \tilde{\mathcal{F}}} \lambda_e c^0(e, x) = T \mathbb{E}_{e \in \tilde{\mathcal{F}}} [c^0(e, x) c^1(e)]$ . Similarly, for any  $y \in f^{-1}(1)$ ,  $C^1(\tilde{\mathcal{F}}, y) = \sum_{e \in \tilde{\mathcal{F}}} p_y(e)^2 c^1(e, y) / \lambda_e \leq 1$ , since terms in the sum are positive only for edges with positive flow. ◀

### 3.3 Loading sparse inputs

We study a particular type of super edges, that we will use repeatedly in the sequel. To construct a learning graph for a given function, one often needs to load a subset  $S$  of the labels. This can be done by a path of length  $|S|$  with negative and positive complexities  $|S|$ , which, after some rebalancing, leads directly to the following lemma.

► **Lemma 3.7.** *For any set  $S$ , there exists a super edge denoted  $\text{DenseLoad}_S$  loading  $S$  with the following complexities for any input  $z \in \{0, 1\}^N$ :*

$$c^0(\text{DenseLoad}_S, z) = |S|^2 \quad \text{and} \quad c^1(\text{DenseLoad}_S, z) = 1.$$

When the input is sparse one can do significantly better as we describe now, where  $|z_S|$  denotes the Hamming weight of  $z_S$ .

► **Lemma 3.8.** *For any set  $S$ , there exists a super edge denoted  $\text{SparseLoad}_S$  loading  $S$  with the following complexities for any input  $z \in \{0, 1\}^N$ :*

$$c^0(\text{SparseLoad}_S, z) \leq 6|S|(|z_S| + 1) \log(|S| + 1) \quad \text{and} \quad c^1(\text{SparseLoad}_S, z) \leq 1.$$

**Proof.** Let us assume for simplicity that  $N = |S|$  and  $S = \{1, \dots, N\}$ . We define the learning graph  $\text{SparseLoad}_S$  as the path through edges  $e_1 = (\emptyset, \{1\})$ ,  $e_2 = (\{1\}, \{1, 2\})$ ,  $\dots$ ,  $e_N = (\{1, \dots, N-1\}, S)$ . The weights are defined as, for  $b \in \{0, 1\}$  and  $z \in Z$ ,

$$w_{e_j}^b(z) = \begin{cases} 3 \cdot (|z_{[j-1]}| + 1) \cdot \log(N + 1) & \text{if } z_j = b, \\ 3N \cdot \log(N + 1) & \text{if } z_j = 1 - b, \end{cases}$$

When  $|z| > 0$ , let us denote  $i_0 = 0$ ,  $i_{|z|+1} = N + 1$  and  $(i_k)_{k=1, \dots, |z|}$  the increasing sequence of indices  $j$  such that  $z_j = 1$ . Then, for  $k = 1, \dots, |z| + 1$ , we define  $m_k$  as the number of indices  $j \in (i_{k-1}, i_k)$  such that  $z_j = 0$ . More precisely,  $m_k = i_k - i_{k-1} - 1$  for  $1 \leq k \leq |z|$  and  $m_{|z|+1} = N - i_{|z|}$ . So  $\sum_{k=1}^{|z|+1} m_k = N - |z|$ . Then, for any input  $z$ ,

$$C^0(\text{SparseLoad}_S, z) = \begin{cases} 3N \cdot \log(N + 1) & \text{if } |z| = 0, \\ 3 \cdot \left( |z|N + \sum_{i=1}^{|z|+1} i \times m_i \right) \cdot \log(N + 1) & \text{otherwise,} \end{cases}$$

which is bounded above by  $6N \cdot (|z| + 1) \cdot \log(N + 1)$ . Moreover, using  $\sum_{i=1}^{|z|+1} \frac{1}{i} \leq \log(|z| + 1) + 1$ , we get

$$C^1(\text{SparseLoad}_S, z) = \frac{1}{3 \cdot \log(N + 1)} \left( (N - |z|) \frac{1}{N} + \sum_{i=1}^{|z|+1} \frac{1}{i} \right) \leq 1. \quad \blacktriangleleft$$

## 4 Composition of learning graphs

To simplify our presentation, we will use the term *empty transition* for an edge between two vertices representing the same set. They carry zero flow and weight, and they do not contribute to any complexity.

### 4.1 Learning graph for OR

Consider  $n$  Boolean functions  $f_1, \dots, f_n$  with respective learning graphs  $\mathcal{G}_1, \dots, \mathcal{G}_n$ . The following lemma explains how to design a learning graph  $\mathcal{G}_{\text{OR}}$  for  $f = \bigvee_{i \in [n]} f_i$  whose complexity is the squared mean of former ones. We will represent  $\mathcal{G}_{\text{OR}}$  graphically as

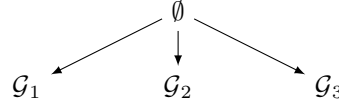
$$\emptyset \xrightarrow{i} \mathcal{G}_i$$

This result is similar to the one of [2], where a search procedure is designed for the case of variable query costs, or equivalently for a search problem divided into subproblems with variable complexities.

► **Lemma 4.1.** *Let  $\mathcal{G}_1, \dots, \mathcal{G}_n$  be learning graphs for Boolean functions  $f_1, \dots, f_n$  over  $Z$ . Assume further that for every  $x$  such that  $f(x) = 1$ , there is at least  $k$  functions  $f_i$  such that  $f_i(x) = 1$ . Then there is a learning graph  $\mathcal{G}$  for  $f = \bigvee_{i \in [n]} f_i$  such that for every  $z \in Z$*

$$\begin{cases} C^0(\mathcal{G}, z) \leq \frac{n}{k} \times \mathbb{E}_{i \in [n]} (C^0(\mathcal{G}_i, z) C^1(\mathcal{G}_i)) & \text{when } f(z) = 0, \\ C^1(\mathcal{G}, z) \leq 1 & \text{when } f(z) = 1. \end{cases}$$

**Proof.** We define the new learning graph  $\mathcal{G}$  by considering a new root  $\emptyset$  that we link to the roots of each  $\mathcal{G}_i$ . In particular, each  $\mathcal{G}_i$  lies in a different connected component. For  $n = 3$ , the graph is displayed below:



Then, we rescale the original weights of edges in each component  $\mathcal{G}_i$  by  $\lambda_i = C^1(\mathcal{G}_i)/k$ . The complexity  $C^0(\mathcal{G}, x)$  for a negative instance  $x$  is

$$C^0(\mathcal{G}, x) = \sum_{i=1}^n \lambda_i C^0(\mathcal{G}_i, x) = \frac{n}{k} \times \mathbb{E}_i (C^0(\mathcal{G}_i, x) C^1(\mathcal{G}_i)).$$

Consider now a positive instance  $y$ . Then  $y$  is also a positive instance for at least  $k$  functions  $f_i$ . Without loss of generality assume further that these  $k$  functions are  $f_1, f_2, \dots, f_k$ . We define the flow of  $\mathcal{G}$  (for  $y$ ) as a flow uniformly directed from  $\emptyset$  to  $\mathcal{G}_i$  for  $i = 1, 2, \dots, k$ . In each component  $\mathcal{G}_i$ , the flow is then routed as in  $\mathcal{G}_i$ . Therefore we have

$$C^1(\mathcal{G}, y) = \sum_{i=1}^k \frac{1}{k^2} \times \frac{C^1(\mathcal{G}_i, y)}{\lambda_i} \leq 1.$$

Finally, observe that by construction the flow is directed to sinks having 1-certificates, thus  $\mathcal{G}_{\text{OR}}$  indeed computes  $f = \bigvee_{i \in [n]} f_i$ . ◀

## 4.2 Learning graph for Johnson walks

We build a framework close to the one of quantum walk based algorithms from [19, 18] but for extended learning graphs. To avoid confusion we encode into a partial assignment the corresponding assigned location, that is,  $z_S = \{(i, z_i) : i \in S\}$ .

Fix some parameters  $r \leq k \leq n$ . We would like to define a learning graph  $\mathcal{G}_{\text{Johnson}}$  for  $f = \bigvee_A f_A$ , where  $A$  ranges over  $k$ -subsets of  $[n]$  and  $f_A$  are Boolean functions over  $Z$ , but differently than in Lemma 4.1. For this, we are going to use a learning graph for  $f_A$  when the input has been already partially loaded, that is, loaded on  $I(A)$  for some subset  $I(A) \subseteq [N]$  depending on  $A$  only. Namely, we assume we are given, for every partial assignment  $\lambda$ , a learning graph  $\mathcal{G}_{A, \lambda}$  defined over inputs  $Z_\lambda = \{z \in Z : z(I(A)) = \lambda\}$  for  $f_A$  restricted to  $Z_\lambda$ .

Then, instead of the learning graph of Lemma 4.1, our learning graph  $\mathcal{G}_{\text{Johnson}}$  factorizes the load of input  $z$  over  $I(A)$  for  $|A| = k$  and then uses  $\mathcal{G}_{A, z_{I(A)}}$ . This approach is more efficient when, for every positive instance  $y$ , there is a 1-certificate  $I(T_y)$  for some  $r$ -subset  $T_y$ , and  $A \mapsto I(A)$  is monotone. This is indeed the analogue of a walk on the Johnson Graph.

We will represent the resulting learning graph  $\mathcal{G}_{\text{Johnson}}$  graphically using  $r + 1$  arrows: one for the first load of  $(k - r)$  elements, and  $r$  smaller ones for each of the last  $r$  loads of a single element. For example, when  $r = 2$  we draw:

$$\emptyset \xrightarrow{A} \mathcal{G}_{A, x_{I(A)}}$$

In the following,  $\text{Load}_S$  denotes any super edge loading the elements of  $S$ , such as  $\text{DenseLoad}$  or  $\text{SparseLoad}$  that we have defined in Lemmas 3.7 and 3.8.

► **Theorem 4.2.** *For every subset  $S \subseteq [N]$ , let  $\text{Load}_S$  be any super edge loading  $S$  with  $c^1(\text{Load}_S) \leq 1$ . Let  $r \leq k \leq n$  and let  $f = \bigvee_A f_A$ , where  $A$  ranges over  $k$ -subsets of  $[n]$  and  $f_A$  are Boolean functions over  $Z$ .*

*Let  $I$  be a monotone mapping from subsets of  $[n]$  to subsets of  $[N]$  with the property that, for every  $y \in f^{-1}(1)$ , there is an  $r$ -subset  $T_y \subseteq [n]$  whose image  $I(T_y)$  is a 1-certificate for  $y$ .*

*Let  $\mathbf{S}, \mathbf{U} > 0$  be such that every  $x \in f^{-1}(0)$  satisfies*

$$\mathbb{E}_{A' \subseteq [n]: |A'|=k-r} (C^0(\text{Load}_{I(A')}, z)) \leq \mathbf{S}^2; \tag{1}$$

$$\mathbb{E}_{A' \subseteq A'' \subseteq [n]: |A'|=|A''|-1=i} (C^0(\text{Load}_{I(A'') \setminus I(A')}, z)) \leq \mathbf{U}^2, \quad \text{for } k-r \leq i < k. \tag{2}$$

*Let  $\mathcal{G}_{A, \lambda}$  be learning graphs for functions  $f_A$  on  $Z$  restricted to inputs  $Z_\lambda = \{z \in Z : z(I(A)) = \lambda\}$ , for all  $k$ -subsets  $A$  of  $[n]$  and all possible assignments  $\lambda$  over  $I(A)$ . Let finally  $\mathbf{C} > 0$  be such that every  $x \in f^{-1}(0)$  satisfies*

$$\mathbb{E}_{A \subseteq [n]: |A|=k} (C^0(\mathcal{G}_{A, x_{I(A)}}, x) C^1(\mathcal{G}_{A, x_{I(A)}}, f)) \leq \mathbf{C}^2. \tag{3}$$

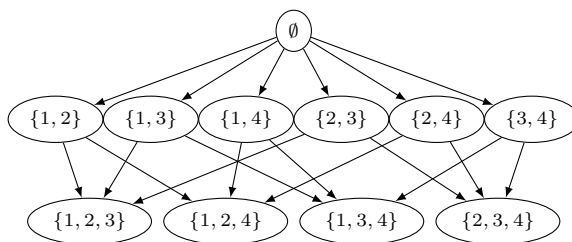
*Then there is a learning graph  $\mathcal{G}_{\text{Johnson}}$  for  $f$  such that for every  $z \in Z$*

$$\begin{cases} C^0(\mathcal{G}_{\text{Johnson}}, z) = O\left(\mathbf{S}^2 + \left(\frac{n}{k}\right)^r (k \times \mathbf{U}^2 + \mathbf{C}^2)\right) & \text{when } f(z) = 0, \\ C^1(\mathcal{G}_{\text{Johnson}}, z) = 1 & \text{when } f(z) = 1. \end{cases}$$

**Proof.**

**Construction.** We define  $\mathcal{G}_{\text{Johnson}}$  by emulating a walk on the Johnson graph  $J(n, k)$  for searching a  $k$ -subset  $A$  having an  $r$ -subset  $T_y$  such that  $I(T_y)$  is a 1-certificate for  $y$ . In that case, by monotonicity of  $I$ , the set  $I(A)$  will be also a 1-certificate for  $y$ .

Our learning graph  $\mathcal{G}_{\text{Johnson}}$  is composed of  $(r + 2)$  stages (that is, layers whose total incoming flow is 1), that we call Stage  $\ell$ , for  $\ell = 0, 1, \dots, r + 1$ . An example of such a learning graph for  $n = 4, k = 3$  and  $r = 1$  is represented below:



Stage 0 of  $\mathcal{G}_{\text{Johnson}}$  consists in  $\binom{n}{k-r}$  disjoint paths, all of same weights, leading to vertices labelled by some  $(k - r)$ -subset  $A'$  and loading  $I(A')$ . They can be implemented by the super edges  $\text{Load}_{I(A')}$ . For positive instances  $y$ , the flow goes from  $\emptyset$  to subsets  $I(A')$  such that  $I(A') \cap T_y = \emptyset$ .

For  $\ell = 1, \dots, r$ , Stage  $\ell$  consists in  $(n - (k - r) - \ell + 1)$  outgoing edges from each node labeled by a  $(k - r + \ell - 1)$ -subset  $A'$ . Those edges are labelled by  $(A', j)$  where  $j \notin A'$  and

load  $I(A' \cup \{j\}) \setminus I(A')$ . They can be implemented by the super edges  $\text{Load}_{I(A' \cup \{j\}) \setminus I(A')}$ . For positive instances  $y$ , for each vertex  $A'$  getting some positive flow, the flow goes out only to the edge  $(A', j_\ell)$ , with the convention  $T_y = \{j_1, \dots, j_r\}$ .

The final Stage  $(r+1)$  consists in plugging in nodes  $A$  the corresponding learning graph  $\mathcal{G}_{A, x_{I(A)}}$ , for each  $k$ -subset  $A$ . We take a similar approach than in the construction of  $\mathcal{G}_{\text{OR}}$  above. The weights of the edges in each component  $\mathcal{G}_{A, z_{I(A)}}$  are rescaled by a factor  $\lambda_A = C^1(\mathcal{G}_{A, x_{I(A)}}) / \binom{n-r}{k-r}$ . For a positive instance  $y$ , the flow is directed uniformly to each  $\mathcal{G}_{A, y_{I(A)}}$  such that  $T_y \subseteq A$ , and then according to  $\mathcal{G}_{A, y_{I(A)}}$ .

Observe that by construction, on positive inputs the flow reaches only 1-certificates of  $f$ . Therefore  $\mathcal{G}_{\text{Johnson}}$  indeed computes  $f$ .

**Analysis.** Remind that the positive edge-complexity of our super edge  $\text{Load}$  is at most 1.

At Stage 0, the  $\binom{n}{k-r}$  disjoint paths are all of same weights. The flow satisfies the hypotheses of Lemma 3.6 with a speciality of  $O(1)$ . Therefore, using inequality (1), the complexity of this stage is  $O(\mathbf{S}^2)$  when  $f(x) = 0$ , and at most 1 otherwise.

For  $\ell = 1, \dots, r$ , at Stage  $\ell$  consists of  $(n - (k - r) - \ell + 1)$  outgoing edges to each node labeled by a  $(k - r + \ell - 1)$ -subset. Take a positive instance  $y$ . Recall that, for each vertex  $A'$  getting some positive flow, the flow goes out only to the edge  $(A', j_\ell)$ . By induction on  $\ell$ , the incoming flow is uniform when positive. Therefore, the flow on each edge with positive flow is also uniform, and the speciality of the stage is  $O((\frac{n}{k})^\ell \cdot k)$ . Hence, by Lemma 3.6 and using inequality 2, the cost of each such stage is  $O((\frac{n}{k})^\ell \cdot k \cdot \mathbf{U}^2)$ . The dominating term is thus  $O((\frac{n}{k})^r \cdot k \cdot \mathbf{U}^2)$ .

The analysis of the final stage (Stage  $(r+1)$ ) is similar to the proof of Lemma 4.1. For a negative instance  $x$ , the complexity of this stage is:

$$\begin{aligned} \sum_A \lambda_A C^0(\mathcal{G}_{A, x_{I(A)}}, x) &= \frac{\binom{n}{k}}{\binom{n-r}{k-r}} \mathbb{E}_A (C^0(\mathcal{G}_{A, x_{I(A)}}, x) C^1(\mathcal{G}_{A, x_{I(A)}})) \\ &= O\left(\left(\frac{n}{k}\right)^r \times \mathbb{E}_A (C^0(\mathcal{G}_{A, x_{I(A)}}, x) C^1(\mathcal{G}_{A, x_{I(A)}}))\right). \end{aligned}$$

Similarly, when  $f(y) = 1$ , we get a complexity at most 1. ◀

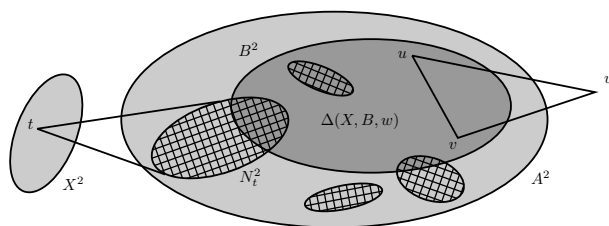
## 5 Application to Triangle Finding

### 5.1 An adaptive Learning graph for dense case

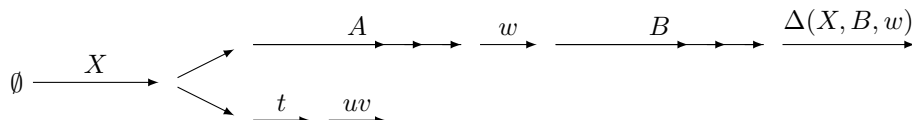
We start by reviewing the main ideas of Le Gall's algorithm in order to find a triangle in an input graph  $G$  with  $n$  vertices. More precisely, we decompose the problem into similar subproblems, and we build up our adaptive learning graph on top of it. Doing so, we get rid of most of the technical difficulties that arise in the resolution of the underlying problems using quantum walk based algorithms.

Let  $V$  be the vertex set of  $G$ . For a vertex  $u$ , let  $N_u$  be the neighborhood of  $u$ , and for two vertices  $u, v$ , let  $N_{u,v} = N_u \cap N_v$ . Figure 1 illustrates the following strategy for finding a potential triangle in some given graph  $G$ , with  $x, a, b$  integer parameters to be specified later.

First, fix an  $x$ -subset  $X$  of vertices. Then, either  $G$  has a triangle with one vertex in  $X$  or each (potential) triangle vertex is outside  $X$ . The first case is quite easy to deal with, so we ignore it for now and we only focus on the second case. Thus there is no need to query any possible edge between two vertices  $u, v$  connected to the same vertex in  $X$ . Indeed, if



■ **Figure 1** Sets involved in Le Gall's algorithm.



■ **Figure 2** Learning graph for Triangle Finding with complexity  $O(n^{5/4})$ .

such an edge exists, the first case will detect a triangle. Therefore one only needs to look for a triangle edge in  $\Delta(X) = \{(u, v) \in V^2 : N_{u,v} \cap X = \emptyset\}$ .

Second, search for an  $a$ -subset  $A$  with two triangle vertices in it. For this, construct the set  $\Delta(X, A) = A^2 \cap \Delta(X)$  of potential triangle edges in  $A^2$ . The set  $\Delta(X, A)$  can be easily set once all edges between  $X$  and  $A$  are known.

Third, in order to decide if  $\Delta(X, A)$  has a triangle edge, search for a vertex  $w$  making a triangle with an edge of  $\Delta(X, A)$ .

Otherwise, search for a  $b$ -subset  $B$  of  $A$  such that  $w$  makes a triangle with two vertices of  $B$ . For this last step, we construct the set  $\Delta(X, B, w) = (N_w)^2 \cap \Delta(X, B)$  of pairs of vertices connected to  $w$ . If any of such pairs is an actual edge, then we have found a triangle.

We will use learning graphs of type  $\mathcal{G}_{\text{OR}}$  for the first step, for finding an appropriate vertex  $w$ , and for deciding whether  $\Delta(X, B, w)$  has an edge; and learning graphs of type  $\mathcal{G}_{\text{Johnson}}$  for finding subsets  $A$  and  $B$ .

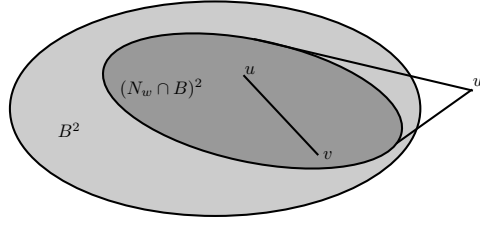
More formally now, let **Triangle** be the Boolean function such that  $\text{Triangle}(G) = 1$  iff graph input  $G$  has a triangle. We do the following decomposition. First, observe that  $\text{Triangle} = \bigvee_{X: |X|=x} (h_X \vee f_X)$  with  $h_X(G) = 1$  (resp.  $f_X(G) = 1$ ) iff  $G$  has a triangle with a vertex in  $X$  (resp. with no vertex in  $X$ ). Then, we pursue the decomposition for  $f_X(G)$  as  $f_X(G) = \bigvee_{A: |A|=a} f_{X,A}(G)$  and  $f_{X,A}(G) = \bigvee_{w \in V} f_{X,A,w}(G)$ , for  $A \subseteq V$  and  $w \in V$ , where

- $f_{X,A}(G) = 1$  iff  $G$  has a triangle between two vertices in  $A \setminus X$  and a third one outside  $X$ ;
- $f_{X,A,w}(G) = 1$  iff  $w \notin X$  and  $G$  has a triangle between  $w$  and two vertices in  $A \setminus X$ .

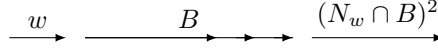
Last, we can write  $f_{X,A,w}(G) = \bigvee_{B \subset A, |B|=b} f_{X,B,w}(G)$ .

With our notations introduced in Section 4, our adaptive learning graph  $\mathcal{G}$  for Triangle Finding can be represented as in Figure 2.

Using adaptive learning graphs instead of the framework of quantum walk based algorithms from [18] simplifies the implementation of the above strategy because one can consider all the possible subsets  $X$  instead of choosing just a random one. Then one only needs to estimate the average complexity over all possible  $X$ . Such an average analysis was not considered in the framework of [18]. In addition, we do not need to estimate the size of  $\Delta(X, A, w)$  at any moment of our algorithm. As a consequence, our framework greatly simplifies the combinatorial analysis of our algorithm as compared to the one of Le Gall, and lets us shave off some logarithmic factors.



■ **Figure 3** Sets involved in the sparse decomposition.



■ **Figure 4** Learning graph for Triangle Finding with complexity  $\tilde{O}((n^{5/6}m^{1/6} + d_2\sqrt{n})\log n)$ .

► **Theorem 5.1.** *The adaptive learning graph of Figure 2 with  $|X| = x$ ,  $|A| = a$ ,  $|B| = b$ , and using Load = DenseLoad, has complexity*

$$O\left(\sqrt{xn^2 + (ax)^2 + \left(\frac{n}{a}\right)^2 \left(a \cdot x^2 + n \left(b^2 + \left(\frac{a}{b}\right)^2 \left(b + \frac{b^2}{x}\right)\right)\right)}\right).$$

In particular, taking  $a = n^{3/4}$  and  $b = x = \sqrt{n}$  leads to  $Q(\text{Triangle}) = O(n^{5/4})$ .

## 5.2 Sparse graphs

In the sparse case we now show to use extended learning graphs in order to get a better complexity than the one of Theorem 5.1.

First, the same learning graph of Theorem 5.1 has a much smaller complexity for sparse graphs when SparseLoad is used instead of DenseLoad.

► **Theorem 5.2.** *The learning graph of Figure 2, using Load = SparseLoad, has complexity over graphs with  $m$  edges*

$$O\left(\sqrt{\left(xm + (ax)^2 \cdot \frac{m}{n^2} + \left(\frac{n}{a}\right)^2 \left(a \cdot x^2 \cdot \frac{m}{n^2} + n \left(b^2 \cdot \frac{m}{n^2} + \left(\frac{a}{b}\right)^2 \left(b + \frac{b^2}{x}\right)\right)\right)}\right) \log n.$$

In particular, taking  $a = n^{3/4}$  and  $b = x = \sqrt{n}/(m/n^2)^{1/3}$  leads to a complexity of  $O(n^{11/12}m^{1/6}\sqrt{\log n})$  when  $m \geq n^{5/4}$ .

We now end with an even simpler learning graph (see Figure 3) whose complexity depends on its average of squared degrees. It consists in searching for a triangle vertex  $w$ . In order to check if  $w$  is such a vertex, we search for a  $b$ -subset  $B$  with an edge connected to  $w$ . For this purpose, we first connect  $w$  to  $B$ , and then check if there is an edge in  $(N_w \cap B)^2$ .

Formally, we do the decomposition  $\text{Triangle} = \bigvee_{w \in V} f_w$ , with  $f_w(G) = 1$  iff  $w$  is a triangle vertex in  $G$ . Then, we pursue the decomposition with  $f_w(G) = \bigvee_{B \subseteq V: |B|=b} f_{w,B}(G)$  where  $f_{w,B}(G) = 1$  iff  $G$  has a triangle formed by  $w$  and two vertices of  $B$ . Using our notations, the resulting learning graph is represented by the diagram in Figure 4.

In the following theorem,  $d_2 = \sqrt{\mathbb{E}_v[|N_v|^2]}$  denotes the variance of the degrees.



► **Theorem 5.3.** *Let  $b \geq n^2/m$ . The learning graph of Figure 4, using SparseLoad for the first stage of  $\mathcal{G}_{\text{Johnson}}$  and DenseLoad otherwise, has complexity over graphs with  $m$  edges*

$$O\left(\sqrt{n\left(b^2\frac{m}{n^2}\log n + \frac{n^2}{b^2}\left(b + \frac{b^2(d_2)^2}{n^2}\right)\right)}\right).$$

Taking  $b = n^{4/3}/(m \log n)^{1/3}$  leads to a complexity of  $O(n^{5/6}(m \log n)^{1/6} + d_2\sqrt{n})$ .

---

## References

- 1 S. Aaronson, S. Ben-David, and R. Kothari. Separations in query complexity using cheat sheets. In *Proceedings of 48th ACM Symposium on Theory of Computing*, pages 863–876, 2016. doi:10.1145/2897518.2897644.
- 2 A. Ambainis. Quantum search with variable times. *Theory of Computing Systems*, 47(3):786–807, 2010. doi:10.1007/s00224-009-9219-1.
- 3 A. Ambainis, K. Balodis, A. Belovs, T. Lee, M. Santha, and J. Smotrovs. Separations in query complexity based on pointer functions. In *Proceedings of 48th ACM Symposium on Theory of Computing*, pages 800–813, 2016. doi:10.1145/2897518.2897524.
- 4 R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001.
- 5 A. Belovs. Learning-graph-based quantum algorithm for  $k$ -distinctness. In *Proceedings of 53rd IEEE Symposium on Foundations of Computer Science*, pages 207–216, 2012.
- 6 A. Belovs. Span programs for functions with constant-sized 1-certificates. In *Proceedings of 44th Symposium on Theory of Computing Conference*, pages 77–84, 2012.
- 7 A. Belovs, A. Childs, S. Jeffery, R. Kothari, and F. Magniez. Time-efficient quantum walks for 3-distinctness. In *Proceedings of 40th International Colloquium on Automata, Languages and Programming*, pages 105–122, 2013.
- 8 A. Belovs and T. Lee. Quantum algorithm for  $k$ -distinctness with prior knowledge on the input. Technical Report arXiv:1108.3022, arXiv, 2011.
- 9 A. Belovs and A. Rosmanis. On the power of non-adaptive learning graphs. In *Proceedings of 28th IEEE Conference on Computational Complexity*, pages 44–55, 2013.
- 10 H. Buhrman, C. Dürr, M. Heiligman, P. Høyer, F. Magniez, M. Santha, and R. de Wolf. Quantum algorithms for element distinctness. *SIAM Journal on Computing*, 34(6):1324–1330, 2005.
- 11 F. Le Gall. Improved quantum algorithm for triangle finding via combinatorial arguments. In *Proceedings of 55th IEEE Foundations of Computer Science*, pages 216–225, 2014. doi:10.1109/FOCS.2014.31.
- 12 F. Le Gall and N. Shogo. Quantum algorithm for triangle finding in sparse graphs. In *Proc. of 26th International Symposium Algorithms and Computation*, pages 590–600, 2015. doi:10.1007/978-3-662-48971-0\_50.
- 13 L. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of 28th ACM Symposium on the Theory of Computing*, pages 212–219, 1996.
- 14 P. Høyer, T. Lee, and R. Špalek. Negative weights make adversaries stronger. In *Proceedings of 39th ACM Symposium on Theory of Computing*, pages 526–535, 2007.
- 15 P. Høyer and R. Špalek. Lower bounds on quantum query complexity. *Bulletin of the European Association for Theoretical Computer Science*, 87, 2005.
- 16 T. Lee, F. Magniez, and M. Santha. Improved quantum query algorithms for triangle finding and associativity testing. *Algorithmica*, 2015. To appear.
- 17 T. Lee, R. Mittal, B. Reichardt, R. Špalek, and M. Szegedy. Quantum query complexity of state conversion. In *Proceedings of 52nd IEEE Symposium on Foundations of Computer Science*, pages 344–353, 2011.

## 20:14 Extended Learning Graphs for Triangle Finding

- 18 F. Magniez, A. Nayak, J. Roland, and M. Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011.
- 19 F. Magniez, M. Santha, and M. Szegedy. Quantum algorithms for the triangle problem. *SIAM Journal on Computing*, 37(2):413–424, 2007.
- 20 N. Nisan. Crew prams and decision trees. *SIAM Journal on Computing*, 20(6):999–1007, 1991. doi:10.1137/0220062.
- 21 B. Reichardt. Reflections for quantum query algorithms. In *Proceedings of 22nd ACM-SIAM Symposium on Discrete Algorithms*, pages 560–569, 2011.
- 22 P. Shor. Algorithms for quantum computation: Discrete logarithm and factoring. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

# Lower Bounds for Elimination via Weak Regularity\*

Arkadev Chattopadhyay<sup>1</sup>, Pavel Dvořák<sup>2</sup>, Michal Koucký<sup>2</sup>,  
Bruno Loff<sup>2</sup>, and Sagnik Mukhopadhyay<sup>5</sup>

- 1 Tata Institute of Fundamental Research, Mumbai, India  
arkadev.c@tifr.res.in
- 2 Charles University, Prague, Czech Republic  
koblich@iuuk.mff.cuni.cz
- 3 Charles University, Prague, Czech Republic  
koucky@iuuk.mff.cuni.cz
- 4 Charles University, Prague, Czech Republic  
bruno@iuuk.mff.cuni.cz
- 5 Tata Institute of Fundamental Research, Mumbai, India  
sagnik@tifr.res.in

---

## Abstract

We consider the problem of elimination in communication complexity, that was first raised by Ambainis et al. [1] and later studied by Beimel et al. [4] for its connection to the famous direct sum question. In this problem, let  $f : \{0, 1\}^{2n} \rightarrow \{0, 1\}$  be any boolean function. Alice and Bob get  $k$  inputs  $x_1, \dots, x_k$  and  $y_1, \dots, y_k$  respectively, with  $x_i, y_i \in \{0, 1\}^n$ . They want to output a  $k$ -bit vector  $v$ , such that there exists one index  $i$  for which  $v_i \neq f(x_i, y_i)$ . We prove a general result lower bounding the randomized communication complexity of the elimination problem for  $f$  using its discrepancy. Consequently, we obtain strong lower bounds for the functions Inner-Product and Greater-Than, that work for exponentially larger values of  $k$  than the best previous bounds.

To prove our result, we use a pseudo-random notion called regularity that was first used by Raz and Wigderson [19]. We show that functions with small discrepancy are regular. We also observe that a weaker notion, that we call weak-regularity, already implies hardness of elimination. Finally, we give a different proof, borrowing ideas from Viola [23], to show that Greater-Than is weakly regular.

**1998 ACM Subject Classification** F.1.1 Models of computation, F.2.2. Analysis of algorithms and Problem Complexity

**Keywords and phrases** communication complexity, elimination, discrepancy, regularity, greater-than

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.21

## 1 Introduction

There is a great interest in computational complexity in so called direct sum questions which consider the complexity of solving  $k$  independent instances of a problem at once. In

---

\* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement n. 616787. The first author is partially supported by a Ramanujan Fellowship of the DST, India and the last author is partially supported by a TCS fellowship.



most reasonable models of computation one would expect the cost of solving  $k$  independent instances at once to be  $k$ -times the cost of solving a single instance. This is indeed true in some models of computation such as decision trees [18, 10]. However, in some computational models we know that one can achieve savings when solving  $k$ -instances simultaneously: in randomized communication complexity [12], in distributional complexity [21] and in zero-error average complexity [15]. For some other models of computations such as boolean circuits we do not know whether some savings are possible. However, in all these cases there is a great interest in the direct sum question as its understanding would shed light on various aspects of complexity. A direct sum theorem for communication complexity would imply the separation of  $NC^1$  from  $NC^2$  [13]. The direct sum for information complexity is used to prove lower bounds in communication complexity (see for example [2, 11, 3, 5]). This motivates the study of the direct sum question.

There are several other problems that are closely related to the direct sum question. In a problem introduced by Beimel et al. [4], one gets  $k$  independent instances of some computational problem and has to decide the correct answer for one of the instances of his choice. Provided that the instances are independent there is a hope of picking some easy instance among the  $k$  instances. Another problem studied in the literature is the problem of selection, where one should select a positive instance among a  $k$  independent instances. This problem was studied exhaustively in the context of structural complexity theory (starting with [20]). Another problem is the problem of distinguishing  $k$  positive instances from  $k$  negative instances [4].

The least difficult among all these problems is the problem of *elimination*. If  $f : \{0, 1\}^{2n} \rightarrow \{0, 1\}$  is a two-player boolean function, the elimination problem  $elim \circ f^k$  gives Alice and Bob  $k$   $n$ -bit strings each, and asks them to find a  $k$ -bit vector  $z$  of answers to the  $k$  instances which differs from the correct answer for at least one of the instances (i.e.  $z_i \neq f(x_i, y_i)$  for some  $i$ ). Hence, one eliminates a single incorrect vector of answers. In the context of communication complexity this problem was posed by Ambainis et al. [1] and further studied by Beimel et al. [4]. To solve the elimination problem one merely has to solve one of the  $k$  instances and negate its answer. Both papers [1, 4] provide examples where one can achieve some savings, typically  $\ll k$  bits of communication, and they also provide lower bounds for particular functions.

Ambainis et al. established  $\Omega(n/\log n \log \log n)$  lower bound on randomized communication complexity of elimination for the Inner-Product –  $elim \circ IP^k$  – and Disjointness –  $elim \circ DISJ^k$  – for constant  $k$ . Their result can be extended to slightly growing  $k$ , but for  $k \geq \Omega(\log n)$  the lower bound becomes trivial. Beimel et al. [4] establish a general relationship, showing that the (public-coin) randomized communication complexity of elimination for  $f^k$  is lower-bounded by the (public-coin) randomized complexity of  $f$  with error roughly  $\frac{1}{2} - 2^{-k}$ . Due to this large allowed error, the lower bound also becomes trivial for large  $k$ .

In this work we also consider the problem of elimination in the setting of (public-coin) randomized communication complexity. We identify two properties of boolean functions that are closely related to the randomized communication complexity of elimination: the *regularity* and *weak-regularity*. Regularity can be thought of as a generalization of discrepancy for functions with non-boolean output. The notion was used by Raz and Wigderson [19] to prove lower bounds in communication complexity. We establish a close relationship between regularity of  $f^k$  and the discrepancy of  $f$  for any function  $f$ . We then relax the notion of regularity to weak-regularity and show that weak-regularity implies lower bounds for the elimination problem. The two results together allow us to lower-bound the randomized communication-complexity of the elimination problem by the inverse-log of discrepancy:

► **Theorem 1.** For any boolean function  $f$  and a distribution  $\mu$  on inputs of  $f$ ,

$$\mathcal{R}^\varepsilon(\text{elim} \circ f^k) \geq \log \frac{1}{\text{Disc}(f)} - \log k + \log(1 - \varepsilon \cdot 2^k) - O(1).$$

One corollary of this theorem is a lower-bound of  $\Omega(n)$  for elimination of  $\text{IP}^k$  for  $k$  as large as exponential in  $n$ . The best known result before our work, due to Beimel et al., does not give any non-trivial lower bound for  $k \geq n$ . Similarly, we show a bound of  $\Omega(\log n)$  for the elimination of  $\text{GT}^k$ , for  $k < n^{1/4}$ , where  $\text{GT}$  is the Greater-Than function. Previous results yielded interesting bounds only for  $k \leq \log n$ .

Our discrepancy to regularity reduction relies on a sophisticated result of Lee et al. [17] which establishes a direct product theorem for discrepancy. For the Greater-Than function  $\text{GT}$  we also provide an elementary proof of weak-regularity. This gives a hope of establishing stronger lower bounds for the elimination of functions that have large discrepancy, such as set disjointness. Our proof of weak-regularity designs a hard distribution for  $\text{GT}$ . Our distribution borrows ideas from the work of Viola [23], but extends them in a novel way. While Viola's distribution is easy for distinguishing  $k$  negative instances from  $k$  positive instances for  $k = O(n)$ , our distribution seems hard even for distinguishing exponentially many positive or negative instances.

## 2 The elimination problem

For a boolean function  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$ , its corresponding *elimination problem*, denoted  $\text{elim} \circ f^k$  for  $k \geq 1$ , is defined as follows: Alice and Bob are given  $k$  strings  $\bar{x} = x_1, \dots, x_k$ , and  $\bar{y} = y_1, \dots, y_k$  respectively, of  $n$ -bits each; they must then communicate in order to agree on an *output* string  $\text{out} \in \{0, 1\}^k$ , such that  $\text{out} \neq f^k(\bar{x}, \bar{y})$  – i.e., they must *eliminate* a single possibility for the value of  $f(x_1, y_1) \dots f(x_k, y_k)$ , from among all possible  $2^k$  values it could take.

The elimination problem was first studied in the context of communication complexity in [1], and later in [4]. Other problems of a similar flavor are studied in those works – enumeration and selection in [1], choice and agreement in [4] – but, in fact, any lower-bounds for elimination will imply the same lower-bounds for these problems. Since proving lower-bounds for elimination will give the same lower-bounds for these other problems, we will not describe the other problems in further detail.

Instead, we will focus on proving lower-bounds against *randomized* protocols for the elimination problem. In this setting, we say that Alice and Bob *succeed* when they output a string  $\text{out} \in \{0, 1\}^k$  other than  $f^k(\bar{x}, \bar{y})$ , as above; otherwise we say that they *made an error*.

► **Definition 2.** The *randomized communication complexity* of  $\text{elim} \circ f^k$ , denoted  $\mathcal{R}^\varepsilon(\text{elim} \circ f^k)$ , is the maximum length of the smallest randomized protocol with shared randomness, which on every input  $\bar{x}, \bar{y}$  will succeed except with *error probability*  $\leq \varepsilon$ .

### 2.1 Basic observations

The first thing to realize is that if both players compute  $f$  for a single instance, then they can output any vector that negates  $f$  at that coordinate. So clearly  $\mathcal{R}^\varepsilon(\text{elim} \circ f^k) \leq \mathcal{R}^\varepsilon(f) \leq \mathcal{D}(f)$ , where  $\mathcal{R}^\varepsilon(f)$  and  $\mathcal{D}(f)$  denote the randomized and deterministic communication complexity of  $f$ , respectively.

The next thing to notice is that the randomized task becomes trivial for  $\varepsilon \geq 2^{-k}$ . Indeed, both players can use their shared randomness to choose a uniformly random  $\text{out} \in \{0, 1\}^k$ ,

## 21:4 Lower Bounds for Elimination via Weak Regularity

and outputting this out causes them to succeed with probability  $1 - 2^{-k}$ . Hence we will assume from this point onward that  $\varepsilon \leq 2^{-k}$ .

As usual, we will make use of Yao's principle to prove our lower-bounds. So let  $\rho \sim \{0, 1\}^{2nk}$  be a distribution, and  $f : \{0, 1\}^{2n} \rightarrow \{0, 1\}$  a boolean function as before.

► **Definition 3.** The *communication complexity* of  $\text{elim} \circ f^k$  over  $\rho$ , denoted  $\mathcal{D}_\rho^\varepsilon(\text{elim} \circ f^k)$ , is the maximum length of the smallest deterministic protocol which succeeds with error probability  $\leq \varepsilon$  on inputs drawn from  $\rho$ .

Yao's principle then tells us that

$$\mathcal{R}^\varepsilon(\text{elim} \circ f^k) = \sup_\rho \mathcal{D}_\rho^\varepsilon(\text{elim} \circ f^k).$$

We will use the easy direction ( $\geq$ ) to prove lower-bounds, by presenting a *hard distribution*  $\rho$  for which  $\mathcal{D}_\rho^\varepsilon(\text{elim} \circ f^k)$  is high.

In preparation to this, let us think what it means when we say  $\mathcal{D}_\rho^\varepsilon(\text{elim} \circ f^k) \leq C$ . It means that we can partition the space  $\{0, 1\}^{nk} \times \{0, 1\}^{nk}$  into  $\leq 2^C$  combinatorial rectangles  $R = A \times B$ , and in each rectangle we will output a single  $\text{out} = \text{out}(R) \in \{0, 1\}^k$ . Now, each rectangle  $R$  is itself naturally partitioned into  $2^k$  *pieces*, one piece for each  $z \in \{0, 1\}^k$ :

$$R^z = \{(\bar{x}, \bar{y}) \in R \mid f^k(x, y) = z\}.$$

(These pieces are possibly empty and non-rectangular.) Then if the success probability is high, it must happen that on the  $\rho$ -large rectangles,  $R^{\text{out}}$  has very little mass. Indeed, the error probability is the sum of every  $\rho(R^{\text{out}(R)})$  over the various rectangles  $R$  induced by the protocol.

This (vague) condition is both necessary and sufficient, because if we do have a (protocol-induced) partition of  $\{0, 1\}^{nk} \times \{0, 1\}^{nk}$  into rectangles, and on every such rectangle  $R$  there is a piece  $R^z$  with very little mass (less than  $2^{-k}\rho(R)$ , say), we may simply output  $\text{out} = z$  on this rectangle, and we will have a non-trivial protocol for elimination.

### 2.2 Regularity

So a natural way of proving a lower-bound would be to show that, under some carefully chosen hard distribution  $\rho$ , every rectangle  $R$  gets split into pieces  $R^z$  all of which are *non-vanishing*. We may eventually come to the following natural definition:

► **Definition 4.** Let  $n, k \geq 1$  be natural numbers, and  $\delta \in [0, 1]$ ; let  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$  be a boolean function, and  $\rho$  be a distribution over  $\{0, 1\}^{nk} \times \{0, 1\}^{nk}$ .

Then  $f$  is said to be  $\delta$ -*weakly-regular* with respect to  $\rho$  if for every  $R$  and  $z$ ,

$$\rho(R^z) \geq 2^{-k}(\rho(R) - \delta),$$

where  $R$  ranges over combinatorial rectangles in  $\{0, 1\}^{nk} \times \{0, 1\}^{nk}$  and  $z$  ranges over  $\{0, 1\}^k$ .

In this definition, if  $\rho(R) < \delta$  then the condition is satisfied trivially so  $\delta$  bounds from below the mass of rectangles for which each  $R^z$  should have non-trivial mass.

Now take any protocol  $\pi$  communicating less than  $C$  bits, and let  $R$  range over the monochromatic rectangles induced by  $\pi$ . Then if  $f$  is  $\delta$ -weakly-regular w.r.t.  $\rho$ ,  $\pi$ 's error probability is

$$\varepsilon = \sum_R \rho(R^{\text{out}(R)}) \geq \sum_R 2^{-k}(\rho(R) - \delta) \geq 2^{-k}(1 - \delta \cdot 2^C).$$

This proves that:

► **Theorem 5.** *If  $f$  is  $\delta$ -weakly-regular with respect to  $\rho$ , then*

$$\mathcal{D}_\rho^\varepsilon(\text{elim} \circ f^k) \geq \log \frac{1 - \varepsilon \cdot 2^k}{\delta}.$$

The notion of *weak* regularity naturally arises from first considerations of the elimination problem. Interestingly, it is the weak version of a stronger notion, which we define below, variants of which first appeared in a paper of Raz and Wigderson [19]<sup>1</sup> on randomized communication complexity of Karchmer–Wigderson games [14], and also later – in disguise – in related work on simulation theorems [9]<sup>2</sup>.

The stronger notion says that  $f$  is *regular* with respect to  $\rho$  if every large rectangle  $R$  is partitioned into rectangles  $R^z$  of roughly equal mass (i.e., each  $R^z$  comprises approximately  $2^{-k}$  fraction of all the  $\rho$ -mass of  $R$ ). Formally:

► **Definition 6.** Let  $n, k, \delta, f$ , and  $\rho$  be as in Definition 4. Then  $f$  is said to be  $\delta$ -regular with respect to  $\rho$  if for every  $R$  and  $z$

$$2^{-k}(\rho(R) - \delta) \leq \rho(R^z) \leq 2^{-k}(\rho(R) + \delta),$$

where  $R$  ranges over combinatorial rectangles in  $\{0, 1\}^{nk} \times \{0, 1\}^{nk}$  and  $z$  ranges over  $\{0, 1\}^k$ .

It is obvious that regularity implies weak regularity. We conjecture that the opposite is not true, i.e., that the two notions can be separated for some function  $f$ . A formal statement of our conjecture appears in the conclusion (§5). In Section 3 we will establish a strong two-way connection between regularity and matrix discrepancy.

The reader may wonder why the notion is called *regularity*. Indeed, if we think of a bipartite graph  $G_z$  with  $\{0, 1\}^{nk}$  on each side, and put an edge between  $\bar{x}$  and  $\bar{y}$  if  $f^k(\bar{x}, \bar{y}) = z$ , then the regularity property implies that the edge-density between any two large sets  $A$  and  $B$  will be roughly what one would expect if the edges of  $G_z$  had been chosen at random. This is regularity in the sense of Szemerédi [22].

## Regularity is a form of pseudo-randomness

We now give an alternative definition for regularity, which we found curious and worth noting. Say that a set  $\mathcal{G} \subset \{0, 1\}^{nk} \times \{0, 1\}^{nk}$  is  $\delta$ -pseudorandom against combinatorial rectangles under  $\rho$  if for any such rectangle  $R \subset \{0, 1\}^{nk} \times \{0, 1\}^{nk}$  we have  $\rho(R) - \delta \leq \frac{\rho(R \cap \mathcal{G})}{\rho(\mathcal{G})} \leq \rho(R) + \delta$ . Call  $\mathcal{G}$  a  $\delta$ -hitting set for combinatorial rectangles under  $\rho$  if  $\frac{\rho(R \cap \mathcal{G})}{\rho(\mathcal{G})} \geq \rho(R) - \delta$ .

Now suppose that  $\rho = \mu^k$ , where  $\mu$  is a balanced distribution over  $\{0, 1\}^n$ . It then follows that  $f$  is  $\delta$ -regular with respect to  $\rho$  if and only if, for every  $z \in \{0, 1\}^k$ ,  $(f^k)^{-1}(z)$  is  $\delta$ -pseudorandom against combinatorial rectangles under  $\rho$ . In the same way,  $f$  is weakly regular if every inverse image  $(f^k)^{-1}(z)$  is a  $\delta$ -hitting set

It should also be clear that, if  $\delta$  is small enough and  $\pi$  is a sufficiently-short protocol, or part of a protocol, then whatever output or transcript distribution  $\pi$  may have on  $\rho$ , the corresponding distribution on the conditional  $\rho \mid (f^k)^{-1}(z)$  will be close, so, in some sense,  $\pi$  cannot distinguish the two distributions. We found this to be a good intuition, although in our results it is used only implicitly.

<sup>1</sup> See the comments on p. 10, before Lemma 2.2, of the full (preprint) version.

<sup>2</sup> When the authors show that block-wise density implies uniformity, they are essentially showing that large  $R$  are broken into equally sized  $R^z$ .

### 2.3 The contents of our paper

The next two sections form the core of the paper. In Section 3 we will explain the relationship between discrepancy and regularity, and this will give us the strongest possible randomized lower-bounds for the elimination problem of any function with sufficiently high discrepancy. This result improves the bound from [4] (we will see how in Section 3), and establishes that regularity holds for a large class of functions, a result which is interesting in its own right.

We then present, in Section 4, a proof of weak regularity of the Greater-Than function. Braverman and Weinstein [6] have proven that the Greater-Than function has small discrepancy ( $O(\frac{1}{\sqrt{n}})$ ), and this, together with the previous Section 3, is enough to prove a comparable lower-bound. However, we feel that this proof is interesting for two reasons. First of all, it is a proof from first principles, whereas the proof that discrepancy implies regularity uses the XOR lemma for discrepancy of Lee et al. [17], which can be considered heavy machinery. Second, perhaps more importantly, it is noteworthy that we are still unable to prove elimination lower-bounds for functions with large discrepancy, such as disjointness. Because regularity is a stronger property than discrepancy, there is little hope to use the regularity property to prove such lower-bounds. But because our second proof is based on weak regularity, we can hope that the techniques therein will one day allow us to understand the elimination problem for high-discrepancy functions.

## 3 Discrepancy and regularity

For the remainder of this section, let  $n, k \geq 1$  be natural numbers; let  $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$  be a boolean function, and  $\mu$  be a distribution over  $\{0, 1\}^n \times \{0, 1\}^n$ ; let the letter  $R$  always denote a combinatorial rectangle in  $\{0, 1\}^{nk} \times \{0, 1\}^{nk}$  and let  $z$  always denote an element of  $\{0, 1\}^k$ . For a given  $R$ , let  $R^z$  be as defined in Section 2:

$$R^z = \{(\bar{x}, \bar{y}) \in R \mid \forall i f(x_i, y_i) = z_i\}.$$

Begin by recalling the well-known notion of matrix discrepancy [16]:

► **Definition 7.** The *discrepancy* of  $f$  with respect to  $\mu$  equals

$$\text{Disc}_\mu(f) = \max_{A \subset \{0,1\}^n, B \subset \{0,1\}^n} \left| \sum_{a \in A, b \in B} \mu(a, b) (-1)^{f(a,b)} \right|.$$

We will now show that discrepancy and regularity (Definition 6) are closely related. On one hand, it is easy to see that regularity for  $k = 1$  is equivalent to discrepancy. To see this, notice that  $f$  is  $\delta$ -regular with respect to  $\mu$  if and only if  $\mu(R^0)$  and  $\mu(R^1)$  are both within  $\frac{\mu(R)}{2} - \frac{\delta}{2}$  and  $\frac{\mu(R)}{2} + \frac{\delta}{2}$  for all  $R$ , which again happens if and only if  $\text{Disc}_\mu(f) \leq \delta$ .

► **Lemma 8.** *Let  $k = 1$ . Then  $f$  is  $\delta$ -regular with respect to  $\mu$  if and only if  $\text{Disc}_\mu(f) \leq \delta$ .*

It is somewhat harder to show the relation for  $k > 1$ . Our proof makes use of the following remarkable result, which was proven in [17, Theorem 19]:

► **Lemma 9** (XOR-lemma for discrepancy). *Let  $\mu^t$  be the  $t$ -fold product of  $\mu$ , and  $\oplus_t f$  be the  $t$ -fold XOR of  $f$ . Then*

$$\text{Disc}_{\mu^t}(\oplus_t f) \leq 64^t \cdot \text{Disc}_\mu(f)^t$$

Our main result in this section is the following:



► **Lemma 10.** *Every  $f$  is  $O(k \cdot \text{Disc}_\mu(f))$ -regular with respect to  $\mu^k$ , whenever  $k \leq \frac{1}{64 \text{Disc}_\mu(f)}$ .*

The constant hidden in the  $O$ -notation is  $64e$  ( $e$  is Euler's number).

**Proof.** Abbreviate  $\rho \equiv \mu^k$ . For any given rectangle  $R = A \times B \subset \{0, 1\}^{nk} \times \{0, 1\}^{nk}$ , we will compute  $\rho(R^z)$  directly. Let  $a$  range over  $A$  and  $b$  over  $B$ , and set  $f_j = f$  when  $z_j = 1$  and  $f_j = 1 - f$  when  $z_j = 0$ ; then

$$\rho(R^z) = \sum_{a,b} \rho(a,b) \prod_{j \in [k]} \frac{1 + (-1)^{f_j(a_j, b_j)}}{2};$$

Expanding the product and separating out the resulting “1” term:

$$\rho(R^z) = 2^{-k} \left( \sum_{a,b} \rho(a,b) + \sum_{\emptyset \neq T \subset [k]} \sigma_T \right),$$

$$\sigma_T = \sum_{a,b} \rho(a,b) \prod_{j \in T} (-1)^{f_j(a_j, b_j)}$$

The left term is simply  $\rho(R)$ , so we now bound  $|\sigma_T|$ . Say  $|T| = t$ ; let  $a' \in \{0, 1\}^{n(k-t)}$  range over the projection<sup>3</sup>  $A_{[k] \setminus T}$  and let  $a''$  range over the elements of  $A_T$  such that  $a'a'' \in A$ ; similarly for  $b'$  and  $b''$  with respect to  $B$ ; then

$$|\sigma_T| \leq \sum_{a',b'} \mu^{k-t}(a', b') \cdot \left| \sum_{a'',b''} \mu^t(a'', b'') \prod_{j \in T} (-1)^{f(a''_j, b''_j)} \right|.$$

This inequality follows from the triangle inequality, since  $\rho = \mu^k$ . Notice that  $f_j$  may be replaced by  $f$ , since any resulting multiplication with  $-1$  gets absorbed by taking the absolute value. Now the innermost sum is upper-bounded by  $\text{Disc}_{\mu^t}(\oplus_t f)$ . Let  $D = 64 \cdot \text{Disc}_\mu(f)$ . Now Lemma 9 allows us to simplify:

$$|\sigma_T| \leq \sum_{a',b'} \mu^{k-t}(a', b') \cdot D^t \leq D^t.$$

But then

$$\sum_{\emptyset \neq T \subset [k]} |\sigma_T| \leq \sum_{t=1}^k \binom{k}{t} D^t = (1 + D)^k - 1.$$

By taking the derivative with respect to  $D$ , we find that  $(1 + D)^k - 1 \leq ekD$  whenever  $k \leq \frac{1}{D}$ . So we conclude that

$$\rho(R) \cdot 2^{-k} - ekD \cdot 2^{-k} \leq \rho(R^z) \leq \rho(R) \cdot 2^{-k} + ekD \cdot 2^{-k}. \quad \blacktriangleleft$$

<sup>3</sup>  $A_S$  for  $S \subset [k]$  is the projection of  $A$  into the coordinates in  $S$ , i.e., those settings  $a'$  of the coordinates in  $S$  which can be completed with some  $a''$  in the coordinates in  $[k] \setminus S$  to get a string in  $A$ .

### Lower-bounds from discrepancy

The connection between regularity and discrepancy results in a general lower-bound for the elimination problem of small-discrepancy functions:

► **Corollary 11.** *For any boolean function  $f : \{0, 1\}^{2n} \rightarrow \{0, 1\}$ , any distribution  $\mu$  over  $\{0, 1\}^{2n}$ , and any  $k \leq \frac{1}{64 \text{Disc}_\mu(f)}$ ,*

$$\mathcal{D}_{\mu^k}^\varepsilon(\text{elim} \circ f^k) \geq \log \frac{1}{\text{Disc}_\mu(f)} - \log k + \log(1 - \varepsilon \cdot 2^k) - O(1).$$

From Corollary 11, Yao’s principle now gives us Theorem 1 mentioned in the introduction. This result should be compared with a similar result, which is implicit in [4]:

► **Proposition 12** ([4]). *For any  $f$  and  $\mu$ ,*

$$\mathcal{D}_{\mu^k}^\varepsilon(\text{elim} \circ f^k) \geq \log \frac{1}{\text{Disc}_\mu(f)} - k + \log(1 - \varepsilon \cdot 2^k) - O(1).$$

As the reader may see, the gain that we achieve is to show the theorem for much larger  $k$ . Our result will still hold for an exponentially larger  $k$  than what was previously allowed.

Recall the *Greater-Than* function  $\text{GT}_n - \text{GT}_n(x, y) = 1$  if and only if  $x \geq y$ , where  $x$  and  $y$  are two  $n$ -bit numbers written in base 2. Braverman and Weinstein [6] have provided a distribution  $\mu$  on which  $\text{Disc}_\mu(\text{GT}_n) = \Omega(\frac{1}{\sqrt{n}})$ , and so as a corollary we obtain the following lower-bound:

► **Corollary 13.**  $\mathcal{R}^\varepsilon(\text{elim} \circ \text{GT}_n^k) \geq \frac{1}{2} \log n - \log k + \log(1 - \varepsilon \cdot 2^k)$ .

When  $\text{IP}_n$  equals the Inner-Product mod-2 function, we get:

► **Corollary 14.**  $\mathcal{R}^\varepsilon(\text{elim} \circ \text{IP}_n^k) \geq \frac{n}{2} - \log k + \log(1 - \varepsilon \cdot 2^k)$ .

While our improvement may seem minor at first, we believe it is actually very significant. Notice that, remarkably, the Inner-Product lower-bound is linear even for  $k = 2^{\Omega(n)}$ . For the Greater-Than function, the previously known lower-bounds would be meaningless for any  $k \geq \log n$ , whereas our lower-bounds can go as far as  $k = \Omega(n)$ . We conjecture that these lower-bounds are optimal when the allowed error is  $\Omega(2^{-k})$ ; we will have more to say in the conclusion.

## 4 Lower-bound for $\text{elim} \circ \text{GT}$ from first principles

Our “hard” distribution  $\mu$  on  $\{0, 1\}^n \times \{0, 1\}^n$  is as follows. Let  $m$  and  $\ell$  be integers such that  $n = m\ell$ . (Think of  $m = \sqrt{n}$ .) We split each  $n$ -bit output into  $m$  blocks of  $\ell$  bits each. We set  $X = X_1 \dots X_m$ , where each block  $X_i$  is uniformly and independently selected from the set  $\mathcal{B}_\ell = \{0, 1\}^\ell - \{0^\ell, 1^\ell\}$  (i.e. we forbid the all-0s and all-1s strings). Then we pick a uniformly-random block-index  $J \in [m]$ , and a uniformly-random bit  $Z \in \{-1, 1\}$ , and set  $Y = X_1 \dots X_{J-1}(X_J + Z)0 \dots 0$ , where  $X_J$  is interpreted as an integer. Let  $\bar{\mu}$  denote the distribution of  $(X, Y, Z, J)$  generated by this process; then  $\mu$  is the projection of  $\bar{\mu}$  onto  $(X, Y)$ .

Then the main theorem of this section is:

► **Theorem 15.**  $\text{GT}_n$  is  $n^{-1/17}$ -weakly-regular with respect to  $\mu^k$ , provided  $k \leq n^{1/4}$ .

Our proof is inspired by a paper of Viola [23] who proved lower bound on the randomized communication complexity of  $\text{GT}_n$ . To prove Theorem 15 we will review some basic lemmas and definitions.

If  $A$  and  $B$  are two random variables over the same universe, we will use  $\Delta(A; B)$  to denote the *statistical distance* (or *total variation distance*) of their distributions; we use  $H(A)$  to denote the *entropy* of  $A$ 's distribution. Definitions of these concepts may be found on [7], or via a simple internet search.

► **Lemma 16** (Pinsker's inequality). *Let  $V$  be a random variable taking values in a set  $S$ , and let  $U$  be a uniform variable over  $S$ . Then  $\Delta(V; U) \leq \sqrt{\log |S| - H(V)}$ .*

See [8, p. 44] for a proof of the above.

► **Lemma 17.** *For  $x \geq 2$ , it holds  $\log(2^x - 2) \geq x - \frac{1}{2^{x-2}}$ .*

**Proof.** We will prove an equivalent inequality

$$\log(2^x - 2) - \log(2^x) \geq -\frac{1}{2^{x-2}}.$$

By convexity of the exponential function we have  $1 - y \geq 2^{-2y}$  for  $y \in [0; \frac{1}{2}]$ . Then

$$\log(2^x - 2) - \log(2^x) = \log(1 - 2^{1-x}) \geq \log(2^{-2^{2-x}}) = -\frac{1}{2^{x-2}}. \quad \blacktriangleleft$$

**Proof of Theorem 15.** Let the random variables  $X = X^1 \dots X^k$ ,  $Y = Y^1 \dots Y^k$ ,  $J = J^1 \dots J^k$  and  $Z = Z^1 \dots Z^k$  be drawn according to the distribution  $\bar{\mu}^k$ , by the process given above, so that  $(X, Y)$  is distributed according to  $\mu^k$ .

Fix a large rectangle  $R = R_1 \times R_2$  - i.e., a rectangle such that  $\mu^k(R) \geq \frac{1}{n}$ . Let  $\mathcal{X} = (\mathcal{B}_\ell)^{mk}$  be the support of  $X$ . Since  $\mu^k$  has zero mass outside of  $\mathcal{X} \times \{0, 1\}^{kn}$ , assume without loss of generality that  $R_1 \subset \mathcal{X}$ .

Let  $W$  denote a random variable distributed as  $Y$  conditioned on  $X \in R_1$ , and  $W_z$  be distributed as  $Y$  conditioned on  $X \in R_1$  and  $Z = z$ . We will prove that, for any  $z \in \{0, 1\}^k$ ,

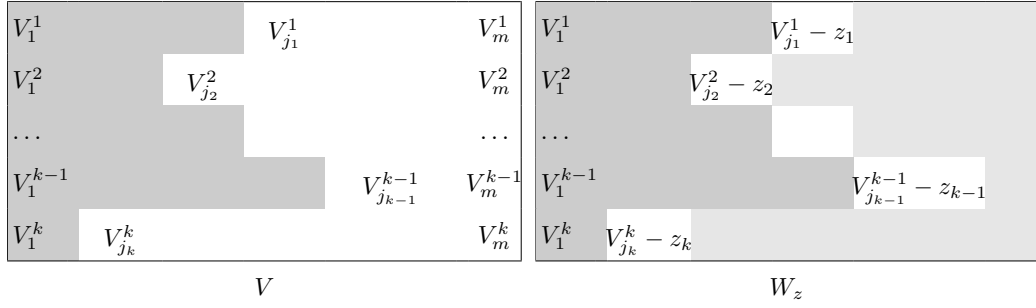
$$\Delta(W; W_z) \leq \frac{1}{n^{1/17}}. \quad (I)$$

This implies Theorem 15, because<sup>4</sup>

$$\begin{aligned} 2^k \mu^k(R^z) &= 2^k \cdot \Pr_{\mu^k}[X \times Y \in R^z] \\ &= 2^k \cdot \Pr_{\mu^k}[X \in R_1, Y \in R_2, Z = z] \\ &= \Pr_{\mu^k}[X \in R_1] \cdot \Pr_{\mu^k}[Y \in R_2 \mid X \in R_1, Z = z] \\ &\geq \Pr_{\mu^k}[X \in R_1] \cdot \left( \Pr_{\mu^k}[Y \in R_2 \mid X \in R_1] - \frac{1}{n^{1/17}} \right) \\ &\geq \Pr_{\mu^k}[X \in R_1] \cdot \Pr_{\mu^k}[Y \in R_2 \mid X \in R_1] - \frac{1}{n^{1/17}} \\ &= \mu^k(R) - \frac{1}{n^{1/17}} \end{aligned}$$

<sup>4</sup> We should note that the same property will trivially hold if  $\mu^k(R) < \frac{1}{n}$ .

21:10 Lower Bounds for Elimination via Weak Regularity



■ **Figure 1**  $V$  and  $W_z$ .  $V_{<j}$  appears in dark gray; light gray marks zeros.

To prove (I), it suffices bounding  $\Delta(W_b; W_a) \leq \frac{1}{n^{1/17}}$  for arbitrary  $a, b \in \{1, -1\}^k$ , because:

$$\begin{aligned} \Delta(W; W_a) &= \frac{1}{2} \sum_{\bar{y}} |\Pr[W = \bar{y}] - \Pr[W_a = \bar{y}]| \\ &= \frac{1}{2} \sum_{\bar{y}} \left| \sum_{b \in \{1, -1\}^k} \Pr[Z = b] \cdot \Pr[W_b = \bar{y}] - \Pr[W_a = \bar{y}] \right| \\ &\leq \sum_{b \in \{1, -1\}^k} \frac{1}{2^k} \cdot \frac{1}{2} \sum_s |\Pr[W_b = \bar{y}] - \Pr[W_a = \bar{y}]| = \sum_{b \in \{1, -1\}^k} \frac{1}{2^k} \Delta(W_b; W_a) \end{aligned}$$

Then let  $V$  denote a random variable distributed as  $X$  conditioned on  $X \in R_1$ ; since  $R_1 \subset \mathcal{X}$  and  $X$  is uniform over  $\mathcal{X}$ ,  $V$  itself is drawn uniformly from  $R_1$ . First we prove that

$$H(V) \geq nk - o(1) - \log n \tag{II}$$

Since  $V$  is uniform on  $R_1$ ,  $H(V) = \log |R_1|$ . As we assumed  $R$  was large,

$$\frac{|R_1|}{|\mathcal{X}|} = \Pr_{(X,Y)}[X \in R_1] \geq \frac{1}{n}.$$

Thus by Lemma 17:

$$H(V) \geq \log \frac{|\mathcal{X}|}{n} = \log [(2^{n/m} - 2)^{mk}] - \log n \geq mk \left( \frac{n}{m} - \frac{1}{2^{n/m} - 2} \right) - \log n = nk - o(1) - \log n.$$

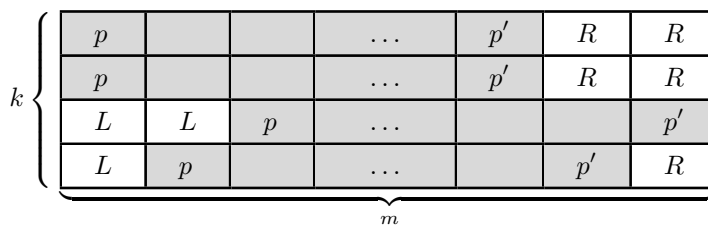
The variable  $V$  can be divided into same blocks as the variable  $X$ . Thus,  $V = V^1, \dots, V^k$  where each  $V^i$  is an  $n$ -bit number. Each  $V^i$  is  $V_1^i \dots V_m^i$  where each  $V_j^i$  is an  $\ell$ -bit number. Let  $j$  be a vector in  $[m]^k$ . By  $V_{<j}$  we denote all blocks  $V_s^i$  for  $i \in [k]$  and  $s < j_i$ . The universe of  $V_{<j}$  is denoted by  $\mathcal{V}_{<j}$ . For  $j \in [m]^k$ ,  $v \in \mathcal{V}_{<j}$  and  $z \in \{-1, 1\}^k$ , let  $V_{j,v,z}$  be the random variable  $V_{j_1}^1 - z_1, \dots, V_{j_k}^k - z_k$ , conditioned on  $V_{<j} = v$ . This is the same distribution as the projection of  $Y$ , conditioned on  $J = j, Z = z$  and  $V_{<j} = v$ , onto blocks given by  $j$ . Figure 1 illustrates the notation.

Then from the triangle inequality we get

$$\Delta(W_b; W_a) \leq \sum_{j \in [m]^k} \sum_{v \in \mathcal{V}_{<j}} \Pr_{J,V}[J = j, V_{<j} = v] \Delta(V_{j,v,b}; V_{j,v,a}).$$

Now notice that  $V_{j,v,a} = \pi(V_{j,v,b}) = \pi'(V_j)$  for some permutations  $\pi$  and  $\pi'$  of the domain  $\{0, 1\}^{\frac{nk}{m}}$ ; then for  $U$  uniform over  $\{0, 1\}^{\frac{nk}{m}}$ ,

$$\Delta(V_{j,v,b}; V_{j,v,a}) \leq \Delta(V_{j,v,b}; U) + \Delta(U; V_{j,v,a}) = 2\Delta(U; V_{j_1}^1, \dots, V_{j_k}^k \mid V_{<j} = v).$$



■ **Figure 2** An example how to cover all blocks in the sum  $M$  (gray blocks) and the variable  $L$  and  $R$ . Each rectangle represents a block of  $\frac{n}{m}$  bits.

Pinsker’s inequality then tells us that  $\Delta(V_j; U) \leq \sqrt{\frac{nk}{m} - H(V_{j_1}^1, \dots, V_{j_k}^k | V_{<j} = v)}$ . We may thus bound

$$\begin{aligned} \Delta(W_b; W_a) &\leq \frac{1}{m^k} \sum_{jv} \Pr[V_{<j} = v] \sqrt{\frac{nk}{m} - H(V_{j_1}^1 \dots V_{j_k}^k | V_{<j} = v)} \\ &\leq \sqrt{\frac{1}{m^k} \sum_{jv} \Pr[V_{<j} = v] \left( \frac{nk}{m} - H(V_{j_1}^1 \dots V_{j_k}^k | V_{<j} = v) \right)} \quad (\text{by concavity of } \sqrt{\cdot}) \\ &= \sqrt{\frac{nk}{m} - \frac{1}{m^k} \sum_j H(V_{j_1}^1 \dots V_{j_k}^k | V_{<j})} \quad (\text{by definition of conditional entropy}) \end{aligned}$$

Now we need to bound the sum  $\sum_j H(V_{j_1}^1 \dots V_{j_k}^k | V_{<j})$ . The sum is over all vectors  $j$  in  $[m]^k$ . We will divide the summands into parts that allow us to use the chain rule. We call a vector  $p \in [m]^k$  a *pattern* if  $p$  contains 1 in some coordinate. We denote the set of all patterns by  $\mathcal{P}$ . For a pattern  $p \in \mathcal{P}$  we define a width  $w(p)$  of pattern  $p$  as the maximum of entries of  $p$ :

$$w(p) = \max_{i \in [k]} p_i.$$

Denote the set of all patterns of width  $w$  by  $\mathcal{P}_w$ . For an integer  $s$ ,  $(p + s) = (p_1 + s, p_2 + s, \dots, p_k + s)$ . In this way, we can rewrite the sum of entropies:

$$\sum_{j \in [m]^k} H(V_{j_1}^1 \dots V_{j_k}^k | V_{<j}) = \sum_{w \in [m]} \sum_{p \in \mathcal{P}_w} \sum_{s=0}^{m-w} H(V_{p_1+s}^1 \dots V_{p_k+s}^k | V_{<(p+s)}). \quad (\text{III})$$

Let fix some pattern  $p \in \mathcal{P}_w$  and bound the last sum  $M = \sum_{s=0}^{m-w} H(V_{p_1+s}^1 \dots V_{p_k+s}^k | V_{<(p+s)})$ . Let  $p'$  be a vector such that we add  $m - w$  to every entry of  $p$ , i.e.,  $p' = (p + m - w)$ . Let  $L$  be blocks of  $V$  “to the left” of  $p$  and  $R$  be blocks “to the right” of  $p'$ . Formally,

$$L = V_1^1 \dots V_{p_1-1}^1 \dots V_1^k \dots V_{p_k-1}^k \quad R = V_{p'_1+1}^1 \dots V_m^1 \dots V_{p'_k+1}^k \dots V_m^k$$

The variables  $L$  and  $R$  are chosen in a way that they, together with the blocks used in the sum  $M$ , “cover” all blocks of  $V$ . For a better understanding see Figure 2.

Note that variables  $L$  and  $R$  contains together  $(w - 1)k$  blocks (i.e.,  $\frac{n}{m}(w - 1)k$  bits) independently of the choice of  $p \in \mathcal{P}_w$ . The chain rule then says that  $H(V) = H(L) + M + H(R | V_{\leq p'})$ , and so

$$M = H(V) - H(L) - H(R | V_{\leq p'}) \geq H(V) - \frac{n}{m}(w - 1)k.$$

## 21:12 Lower Bounds for Elimination via Weak Regularity

We are ready to bound the sums from Equation (III).

$$\begin{aligned}
\sum_{w \in [m]} \sum_{p \in \mathcal{P}_w} \sum_{s=0}^{m-w} H(V_{p_1+s}^1 \cdots V_{p_k+s}^k | V_{<(p+s)}) &\geq \sum_{w \in [m]} \sum_{p \in \mathcal{P}_w} H(V) - \frac{n}{m}(w-1)k \\
&\geq \sum_{w \in [m]} \sum_{p \in \mathcal{P}_w} nk - o(1) - \log n - \frac{n}{m}(w-1)k && \text{(by II)} \\
&= \frac{nk}{m} \left( \sum_{w \in [m]} \sum_{p \in \mathcal{P}_w} m - w + 1 \right) - |\mathcal{P}|(o(1) + \log n) \\
&= m^k \frac{nk}{m} - |\mathcal{P}|(o(1) + \log n)
\end{aligned}$$

Bounding the number of all patterns by  $|\mathcal{P}| \leq km^{k-1}$ , we can also bound

$$\begin{aligned}
\Delta(W_a; W_b) &\leq \sqrt{\frac{nk}{m} - \frac{1}{m^k} \sum_j H(V_{j_1}^1 \cdots V_{j_k}^k | V_{<j})} \\
&\leq \sqrt{\frac{nk}{m} - \frac{1}{m^k} \left( m^k \frac{nk}{m} - |\mathcal{P}|(o(1) + \log n) \right)} \\
&= \sqrt{\frac{km^{k-1}}{m^k} (\log n + o(1))} && \text{(for } m = \sqrt{n}, k \leq n^{1/4}\text{)} \\
&= \sqrt{\frac{1}{n^{1/4}} (\log n + o(1))} \leq \frac{1}{n^{1/7}} \quad \blacktriangleleft
\end{aligned}$$

## 5 Conclusion and open problems

We have given strong lower bounds on the elimination problem with an exponentially improved dependence on  $k$  for functions with small discrepancy. We have singled out two measures of complexity, regularity and weak regularity, which appeared implicitly in previous works on communication complexity. We have found regularity to be a natural property to keep in mind, whenever Alice and Bob are given multiple instances  $x_1, y_1, \dots, x_k, y_k$ , and must solve some problem that depends on the pointwise application  $f^k(\bar{x}, \bar{y})$ .

The first question that is not at all clear to us is: how do these notions relate to each other? Is there a function  $f$  for which the elimination problem is hard, and which is still not weakly-regular? Is there a function  $f$  which is weakly-regular but not regular?

**Open problem.** Can we separate the notion of elimination from weak regularity, and weak regularity from regularity?

The second problem we would like to understand is the following. As far as we are able to tell, it could be that elimination is simply as hard as communication of a single instance. We would like to see settled the following conjecture:

**Conjecture (Elimination is as hard as communication)**

$$\mathcal{R}^\varepsilon(\text{elim} \circ f^k) \geq \Omega(\mathcal{R}^\delta(f)) \quad \text{(for } \delta = \Omega(1), \varepsilon = \Omega(2^{-k}) \text{ and } k \leq 2^{\Omega(\mathcal{R}^\delta(f))}\text{)}$$

Third, and finally, we would like to know if our lower-bounds are tight with respect to the parameter  $k$ . We do not know whether this is the case. For example, it could be that if

$k$  is as large as  $10n$ , say, then it would be possible to solve  $\text{elim} \circ \text{GT}^k$  with  $o(\log n)$  bits; or if  $k \geq 2^{10n}$ , say, solving  $\text{elim} \circ \text{IP}_n^k$  would be possible with  $o(n)$  bits of communication. Or it could be that stronger lower-bounds can be proven, with much larger  $k$ . This is an open question.

**Acknowledgements.** Part of the research for this work was done at the Institut Henri Poincaré, as part of the *Nexus of Information and Computation Theories* workshop.

---

## References

- 1 Andris Ambainis, Harry Buhrman, William Gasarch, Bala Kalyanasundaram, and Leen Torenvliet. The communication complexity of enumeration, elimination, and selection. *Journal of Computer and System Sciences*, 63(2):148–185, 2001.
- 2 Ziv Bar-Yossef, T.S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. In *Proceedings of the 43rd Symposium on Foundations of Computer Science (FOCS)*, pages 209–218, 2002.
- 3 Boaz Barak, Mark Braverman, Xi Chen, and Anup Rao. How to compress interactive communication. In *Proceedings of the 42nd Symposium on Theory of Computing (STOC)*, pages 67–76, 2010.
- 4 Amos Beimel, Sebastian Ben Daniel, Eyal Kushilevitz, and Enav Weinreb. Choosing, agreeing, and eliminating in communication complexity. *Computational Complexity*, 23(1):1–42, 2014.
- 5 Mark Braverman and Anup Rao. Information equals amortized communication. In *Proceedings of the 52nd Symposium on Foundations of Computer Science (FOCS)*, pages 748–757, 2011.
- 6 Mark Braverman and Omri Weinstein. A discrepancy lower bound for information complexity. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 459–470. Springer, 2012.
- 7 Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006.
- 8 Imre Csiszar and János Körner. *Information theory: coding theorems for discrete memoryless systems*. Cambridge University Press, 2011.
- 9 Mika Göös, Shachar Lovett, Raghu Meka, Thomas Watson, and David Zuckerman. Rectangles are nonnegative juntas. In *Proceedings of the 47th Symposium on Theory of Computing (STOC)*, pages 257–266. ACM, 2015. 0.
- 10 R. Jain, H. Klauck, and M. Santha. Optimal direct sum results for deterministic and randomized decision tree complexity. *Information Processing Letters*, 110(20):893–897, 2010.
- 11 T. S. Jayram, Ravi Kumar, and D. Sivakumar. Two applications of information complexity. In *Proceedings of the 35th Symposium on Theory of Computing (STOC)*, pages 673–682, 2003.
- 12 M. Karchmer, E. Kushilevitz, and N. Nisan. Fractional covers and communication complexity. *SIAM Journal on Discrete Mathematics*, 8(1):76–92, 1995.
- 13 M. Karchmer, R. Raz, and A. Wigderson. Super-logarithmic depth lower bounds via the direct sum in communication complexity. *Computational Complexity*, 5(3/4):191–204, 1995.
- 14 Mauricio Karchmer and Avi Wigderson. Monotone circuits for connectivity require super-logarithmic depth. *SIAM Journal on Discrete Mathematics*, 3(2):255–265, 1990.
- 15 Gilat Kol, Shay Moran, Amir Shpilka, and Amir Yehudayoff. Direct sum fails for zero-error average communication. In *Proceedings of the 5th Innovations in Theoretical Computer Science (ITCS)*, pages 517–522, 2014.

## 21:14 Lower Bounds for Elimination via Weak Regularity

- 16 Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, New York, NY, USA, 1997. 0.
- 17 Troy Lee, Adi Shraibman, and Robert Špalek. A direct product theorem for discrepancy. In *Proceedings of the 23rd Conference on Computational Complexity (CCC)*, pages 71–80. IEEE, 2008.
- 18 Noam Nisan, Steven Rudich, and Mike Saks. Products and help bits in decision trees. *SIAM Journal on Computing*, 28(3):1035–1050, 1999.
- 19 Ran Raz and Avi Wigderson. Probabilistic communication complexity of boolean relations. In *Proceedings of the 30th Symposium on Foundations of Computer Science (FOCS)*, pages 562–567, 1989.
- 20 Alan L. Selman. P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP. In *Proceedings of the 6th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 546–555, 1979.
- 21 Ronen Shaltiel. Towards proving strong direct product theorems. In *Proceedings of the 16th Conference on Computational Complexity (CCC)*, pages 107–119, 2001.
- 22 Endre Szemerédi. Regular partitions of graphs. Technical report, DTIC Document, 1975.
- 23 Emanuele Viola. The communication complexity of addition. *Combinatorica*, 35(6):703–747, 2015.



# Parameterized and Approximation Results for Scheduling with a Low Rank Processing Time Matrix\*

Lin Chen<sup>1</sup>, Dániel Marx<sup>†2</sup>, Deshi Ye<sup>‡3</sup>, and Guochuan Zhang<sup>§4</sup>

1 Department of Computer Science, University of Houston, Houston, TX, USA  
chenlin198662@gmail.com

2 MTA SZTAKI, Hungarian Academy of Science, Budapest, Hungary  
dmarx@cs.bme.hu

3 Zhejiang University, College of Computer Science, Hangzhou, China  
yedeshi@zju.edu.cn

4 Zhejiang University, College of Computer Science, Hangzhou, China  
zgc@zju.edu.cn

---

## Abstract

We study approximation and parameterized algorithms for  $R||C_{max}$ , focusing on the problem when the rank of the matrix formed by job processing times is small. Bhaskara et al. [2] initiated the study of approximation algorithms with respect to the rank, showing that  $R||C_{max}$  admits a QPTAS (Quasi-polynomial time approximation scheme) when the rank is 2, and becomes APX-hard when the rank is 4.

We continue this line of research. We prove that  $R||C_{max}$  is APX-hard even if the rank is 3, resolving an open problem in [2]. We then show that  $R||C_{max}$  is FPT parameterized by the rank and the largest job processing time  $p_{max}$ . This generalizes the parameterized results on  $P||C_{max}$  [17] and  $R||C_{max}$  with few different types of machines [15]. We also provide nearly tight lower bounds under Exponential Time Hypothesis which suggests that the running time of the FPT algorithm is unlikely to be improved significantly.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics

**Keywords and phrases** APX-hardness, Parameterized algorithm, Scheduling, Exponential Time Hypothesis

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.22

## 1 Introduction

We consider the classical problem of scheduling independent jobs on parallel machines. In this problem, every job  $j$  is required to be processed non-preemptively on one of the machines, and has a processing time  $p_{ij} \in \mathbb{N}$  if it is processed on machine  $i$ . The goal is to assign jobs to machines such that the makespan (maximum job completion time) is minimized.

---

\* A full version of the paper is available at [https://www.researchgate.net/publication/313852592\\_Parameterized\\_and\\_approximation\\_results\\_for\\_scheduling\\_with\\_a\\_low\\_rank\\_processing\\_time\\_matrix](https://www.researchgate.net/publication/313852592_Parameterized_and_approximation_results_for_scheduling_with_a_low_rank_processing_time_matrix).

† Research supported in part by ERC Grant Agreement no. 280152

‡ Research supported in part by NSFC 11271325 and NSFC 11671355.

§ Research supported in part by NSFC 11271325 and NSFC 11671355.



This problem is usually referred to as unrelated machine scheduling (with the objective of makespan minimization), and denoted as  $R||C_{max}$ . Specifically, if  $p_{ij} = p_j/s_i$ , the problem is called uniformly related machine scheduling, and denoted as  $Q||C_{max}$ . Furthermore, if  $p_{ij} = p_j$ , the problem is called identical machine scheduling and denoted as  $P||C_{max}$ .

As we will provide details later, the unrelated machine scheduling problem  $R||C_{max}$  is considerably harder than its special cases  $Q||C_{max}$  and  $P||C_{max}$ . From the perspective of approximation algorithms,  $Q||C_{max}$  admits a PTAS (Polynomial Time Approximation Scheme) [11], while a  $(1.5 - \epsilon)$ -approximation algorithm with  $\epsilon > 0$  being any small constant for  $R||C_{max}$  implies  $P = NP$  [16]. From the perspective of FPT (Fixed Parameter Tractable) algorithms,  $P||C_{max}$  and  $Q||C_{max}$  are FPT parameterized by  $p_{max}$  (the largest job processing time) [15, 17], while  $R||C_{max}$  remains NP-hard even if  $p_{max}$  is 3 [16]. Consequently, various intermediate models are studied in literature, aiming to bridge the way from  $P||C_{max}$  or  $Q||C_{max}$  to  $R||C_{max}$ . Recently, Bhaskara et al. studied the scheduling problem from a new perspective. In their seminal paper [2], they consider the rank of the matrix formed by the processing times of jobs, i.e., the rank of  $P = (p_{ij})_{m \times n}$  where  $m, n$  are the number of machines and jobs, respectively. From this point of view,  $Q||C_{max}$  is the scheduling problem with a matrix of rank 1, while  $R||C_{max}$  is the scheduling problem with a matrix of an arbitrary rank, specifically, the rank may be as large as  $m$ . It thus becomes a very natural question that whether we can find better algorithms for  $R||C_{max}$  if the rank is small.

For simplicity, from now on we call the problem of minimum makespan scheduling with the matrix of processing times that has the rank of  $d$  as rank- $d$  scheduling. It is shown by Bhaskara et al. [2] that rank-2 scheduling admits a QPTAS (Quasi-polynomial Time Approximation Scheme), while rank-4 scheduling becomes APX-hard, leaving open the approximability of rank-3 scheduling.

We continue this line of research in this paper by studying approximation and parameterized algorithms for  $R||C_{max}$  with respect to the rank of the matrix. Our first result is the following theorem, which answers the open problem in [2].

► **Theorem 1.** *Assuming  $P \neq NP$ , for any fixed  $\rho < 2^{-14}$  there does not exist a  $(1 + \rho)$ -approximation algorithm for  $R||C_{max}$ , even if the rank of the matrix formed by job processing times is 3.*

In contrast to the APX-hardness of the rank-3 scheduling, we show that  $R||C_{max}$  is FPT parameterized by  $p_{max}$  and  $d$ .

► **Theorem 2.** *There is an FPT algorithm for  $R||C_{max}$  that runs in  $2^{2^{O(d \log p_{max})}} + n^{O(1)}$  time.*

Notice that  $R||C_{max}$  remains NP-hard even if  $p_{max} = 3$  [16] or  $d = 1$  [8], therefore parameterizing by only  $p_{max}$  or  $d$  does not suffice.

We complement this algorithmic result by the following lower bound.

► **Theorem 3.** *There is no  $2^{2^{O(d \log p_{max})}}$  time algorithm for  $R||C_{max}$ , unless ETH (Exponential Time Hypothesis) fails.*

The approximability of rank  $d$  scheduling is not smooth with respect to the rank  $d$ , as is already observed by Bhaskara et al. [2], yet it is FPT parameterized by  $p_{max}$  and  $d$ , with a running time doubly exponential in  $d$ . Furthermore, such a running time is unlikely to be improved significantly, as is suggested by the lower bound.

We also discuss the possibility of replacing the parameter  $p_{max}$  by  $\bar{p}$ , which is the number of distinct processing times in matrix  $P$ . It is shown by Goemans and Rothvoss [9] that  $P||C_{max}$

is in XP parameterized by  $\bar{p}$ , i.e., there exists a polynomial time algorithm for  $P||C_{max}$  if  $\bar{p}$  is a constant. Indeed, they establish a structural theorem on integer programming, through which we can further show that  $R||C_{max}$  is in XP parameterized by  $\bar{p}$  and  $d$ . It remains as an important open problem whether  $P||C_{max}$  is FPT parameterized by  $\bar{p}$ .

► **Theorem 4.**  $R||C_{max}$  can be solved in  $(\log p_{max})^{2^{O(\zeta)}} + 2^{2^{O(\zeta^2)}} (\log p_{max})^{O(1)}$  time, where  $\zeta = 2^{O(d \log \bar{p})}$ .

**Related work.** Scheduling is a fundamental problem in combinatorial optimization and has received considerable attention in history. In the following we provide a very brief overview with the focus on approximation and parameterized algorithmic results.

In 1988, Hochbaum and Shmoys [11] presented a PTAS for  $P||C_{max}$  as well as  $Q||C_{max}$ . Their algorithm has a running time of  $(n/\epsilon)^{O(1/\epsilon^2)}$ . Subsequent improvements on the running time of the PTAS can be found in [1, 13]. So far, the best PTAS for  $Q||C_{max}$  is due to Jansen, Klein and Verschae [14] and has a running time of  $2^{O(1/\epsilon \log^{O(1)} 1/\epsilon)} + O(n)$ . It is further shown by Chen, Jansen and Zhang [5] that such a running time is essentially the best possible unless ETH fails, even for  $P||C_{max}$ . For the unrelated machine scheduling problem  $R||C_{max}$ , Lenstra, Shmoys and Tardos [16] showed that it does not admit any approximation algorithm with a ratio strictly smaller than 1.5 unless  $P = NP$ . They also provided a 2-approximation algorithm, which was slightly improved to a  $(2 - 1/m)$ -approximation algorithm by Shchepin et al. [20].

A lot of intermediate models between  $R||C_{max}$  and  $Q||C_{max}$  or  $P||C_{max}$  are studied in literature. In this paper, we are most concerned with the rank of the matrix formed by job processing times on machines, i.e., the rank of  $P = (p_{ij})_{m \times n}$ . Bhaskara et al. [2] initiated the study on approximation algorithms for  $R||C_{max}$  with respect to the parameter rank. They showed that rank-2 scheduling admits a QPTAS, while rank-4 scheduling is already APX-hard. Very recently Chen et al. [6] further improves their result by showing that rank-4 scheduling does not admit any approximation algorithm with a ratio that is strictly smaller than 1.5, unless  $P = NP$ .

This new model of scheduling with a small matrix rank is closely related to the problem of scheduling unrelated machines of few different types, which is another intermediate model that receives much study in literature [4, 7, 19, 15]. In the problem of scheduling unrelated machines of few different types, there are  $K$  different types of machines. If two machines  $i$  and  $i'$  are of the same type, then for every job  $j$  it follows that  $p_{ij} = p_{i'j}$ . Simply speaking, machines could be divided into  $K$  disjoint groups such that machines belonging to the same group are identical. It is shown by Bonifaci and Wiese [4] that if  $K$  is a constant, then there exists a PTAS. A PTAS of improved running time was recently presented by Gehrke et al. [19]. It is very easy to see that the problem of scheduling unrelated machines of  $K$  different types is actually a special case of the scheduling problem with a matrix of rank  $K$ .

Compared with the study on approximation algorithms for the scheduling problem, the study on parameterized algorithms is relatively new. Mnich and Wiese [17] were the first to study FPT algorithms for the scheduling problem. They showed that  $P||C_{max}$  is FPT parameterized by  $p_{max}$ , the largest job processing times. Meanwhile  $R||C_{max}$  is FPT parameterized by the number of machines  $m$  and the number of distinct job processing times  $\bar{p}$ . As all job processing times are integers,  $\bar{p}$  is upper bounded by  $p_{max}$ . Hence, their results also imply that  $R||C_{max}$  is FPT parameterized by  $m$  and  $p_{max}$ . Very recently, Knop and Koutecký [15] considered the problem of scheduling unrelated machines of few different types, and showed that  $R||C_{max}$  is FPT parameterized by  $p_{max}$  and  $K$ , where  $K$  is the number

of different types of machines. FPT algorithms for the scheduling problem with different models have also received much study in literature, see, e.g., [3, 22].

It is, however, not clear whether  $R||C_{max}$  is FPT parameterized by  $K$  and  $\bar{p}$ . A recent paper by Goemans and Rothvoss [9] showed that  $P||C_{max}$  could be solved in  $(\log p_{max})^{2^{O(\bar{p})}}$  time. Therefore  $P||C_{max}$  is in XP parameterized by  $\bar{p}$ , i.e., if there is only a constant number of distinct job processing times, then  $P||C_{max}$  could be solved in polynomial time. Indeed, the general structural theorem established in their paper further implies that  $R||C_{max}$  is in XP parameterized by  $K$  and  $\bar{p}$ .

## 2 Preliminaries

Let  $P = (p_{ij})_{m \times n}$  with  $p_{ij} \in \mathbb{N}$  being the processing time of job  $j$  on machine  $i$ . Let  $d$  be the rank of  $P$ . By linear algebra, the matrix  $P$  can be expressed as  $P = MJ$ , where  $M$  is an  $m \times d$  matrix and  $J$  is a  $d \times n$  matrix. We can interpret each row vector  $u_i$  of  $M$  as the  $d$ -dimensional *speed vector* of machine  $i$ , and each column vector  $v_j^T$  of  $J$  as the  $d$ -dimensional *size vector* of job  $j$ . The processing time of job  $j$  on machine  $i$  is then the product of the two corresponding vectors, i.e.,  $p_{ij} = u_i \cdot v_j^T$ . Bhaskara et al. [2] formally define the scheduling problem with low rank processing time matrix by explicitly giving the speed vector of every machine and the size vector of every job. In our paper, we do not necessarily require that the speed and size vectors are given. If these vectors are not given, we take an arbitrary decomposition of the matrix  $P$  into  $P = MJ$ . Therefore, throughout this paper, we do not necessarily require an entry in a speed vector or a size vector to be an integer or a non-negative number.

Some lower bounds on the running time of algorithms in this paper are based on the following Exponential Time Hypothesis (ETH), which was introduced by Impagliazzo, Paturi, and Zane [12]:

**Exponential Time Hypothesis (ETH):** There is a positive real  $\delta$  such that 3SAT with  $n$  variables and  $m$  clauses cannot be solved in time  $2^{\delta n(n+m)^{O(1)}}$ .

Using the Sparsification Lemma by Impagliazzo et al. [12], ETH implies that there is no algorithm for 3SAT with  $n$  variables and  $m$  clauses that runs in time  $2^{\delta m(n+m)^{O(1)}}$  for some  $\delta > 0$  as well.

## 3 APX-hardness for rank-3 scheduling

The whole section is devoted to the proof of Theorem 1. For ease of presentation, when we prove the APX-hardness for rank-3 scheduling, we may construct jobs of fractional processing times. However, by scaling we can easily make all the fractional values into integers.

We start with the one-in-three 3SAT problem, which is a variation of the 3SAT problem. An input of the one-in-three 3SAT problem is a boolean formula that is a conjunction of clauses, where each clause is a disjunction of exactly three 3 literals. The formula is satisfied if and only if there exists a truth assignment of variables such that in every clause there is exactly one true literal, i.e., every clause is satisfied by exactly one variable. It is proved in [18] that it is NP-complete to determine whether an arbitrary given instance of the one-in-three 3SAT problem is satisfiable.

We reduce from a variation of the one-in-three 3SAT problem. Given an instance of the one-in-three 3SAT problem, say,  $I_{sat}$ , we can apply Tovey's method [21] to transform it into  $I'_{sat}$  such that:

- each clause of  $I_{sat}$  contains two or three literals;
- each variable appears three times in clauses. Among its three occurrences there are either two positive literals and one negative literal, or one positive literal and two negative literals;
- there exists a truth assignment for  $I'_{sat}$  where every clause is satisfied by exactly one literal if and only if there is a truth assignment for  $I_{sat}$  where every clause is satisfied by exactly one literal.

The transformation is straightforward. For any variable  $z$ , if it only appears once in the clauses, then we add a dummy clause as  $(z \vee \neg z)$ . Otherwise suppose it appears  $d \geq 2$  times in the clauses, then we replace its  $d$  occurrences with  $d$  new variables as  $z_1, z_2, \dots, z_d$ , and meanwhile add  $d$  clauses as  $(z_1 \vee \neg z_2), (z_2 \vee \neg z_3), \dots, (z_d \vee \neg z_1)$  to enforce that these new variables should take the same truth assignment. It is not difficult to verify that the constructed instance satisfies the above requirements.

Throughout the following part of this section we assume that  $I'_{sat}$  contains  $n$  variables and  $m$  clauses. Let  $\epsilon$  be an arbitrary small positive number. Let  $\tau = 2^3$ ,  $r = 2^{11}\tau = 2^{14}$ ,  $N = n/\epsilon^2$ . We will construct an instance  $I_{sch}$  of the rank-3 scheduling problem such that:

- if there is a truth assignment for  $I'_{sat}$  where every clause is satisfied by exactly one variable, then  $I_{sch}$  admits a feasible schedule whose makespan is  $r + c\epsilon$  for some constant  $c$ ;
- if  $I_{sch}$  admits a feasible schedule whose makespan is strictly less than  $r + 1$ , then there exists a truth assignment for  $I'_{sat}$  where every clause is satisfied by exactly one variable.

We claim that, given the above construction, Theorem 1 follows. To see why, suppose on the contrary that there exists a  $(1 + \rho)$ -approximation algorithm for some constant  $\rho < 2^{-14}$ . We set  $\epsilon = \frac{1-r\rho}{c\rho} = \frac{1-2^{14}\rho}{c\rho}$ , and apply this algorithm to the constructed instance  $I_{sch}$ . There are two possibilities. If  $I'_{sat}$  is satisfiable, then the approximation algorithm returns a feasible solution whose makespan is at most  $(r + c\epsilon)(1 + \rho) = r + r\rho + c\rho \cdot \epsilon < r + 1$ . If  $I'_{sat}$  is not satisfiable, then  $I_{sch}$  does not admit a feasible schedule whose makespan is strictly less than  $r + 1$ , i.e., any feasible schedule has a makespan at least  $r + 1$ , whereas the  $(1 + \rho)$ -approximation algorithm returns a solution whose makespan is at least  $r + 1$ . Thus, we can use the  $(1 + \rho)$ -approximation algorithm to determine the satisfiability of  $I'_{sat}$ , and consequently the satisfiability of  $I_{sat}$  in polynomial time, which contradicts the NP-hardness of the one-in-three 3SAT problem.

**Construction of the scheduling instance.** To construct the scheduling instance, we construct the size vector of every job and speed vector of every machine. Each vector is a triple of three positive numbers. The processing time of a job on a machine is then the inner product of the two corresponding vectors. As we describe in Section 2, the constructed instance is a feasible instance of rank-3 scheduling.

Recall that  $r = 2^{14}$ ,  $\tau = 2^3$ ,  $N = n/\epsilon^2$ . Indeed, if we do not care much about the value of  $\rho$  and only want to show APX-hardness, it suffices to think  $r$  as some value significantly larger than  $\tau$ . For a job  $j$  we denote by  $s(j)$  its size vector.

We construct two main kinds of jobs, element jobs and tuple jobs. In the following we first construct element jobs, which are further divided into variable jobs, truth-assignment jobs, clause jobs and dummy jobs.

- **Variable jobs.** For every variable  $z_i$ , we construct 8 variable jobs,  $v_{i,k}^\gamma$  for  $k = 1, 2, 3, 4$  and  $\gamma = T, F$ . Their size vectors are:

$$s(v_{i,1}^T) = (\epsilon N^{4i+1}, 0, r/8 - 10\tau - 2), \quad s(v_{i,2}^T) = (\epsilon N^{4i+2}, 0, r/8 - 20\tau - 2),$$

$$s(v_{i,3}^T) = (\epsilon N^{4i+3}, 0, r/8 - 18\tau - 2), \quad s(v_{i,4}^T) = (\epsilon N^{4i+4}, 0, r/8 - 12\tau - 2).$$

$$s(v_{i,k}^F) = s(v_{i,k}^T) - (0, 0, 2), k = 1, 2, 3, 4$$

- **Truth-assignment jobs.** For every variable  $z_i$ , we construct eight truth-assignment jobs,  $a_i^\gamma, b_i^\gamma, c_i^\gamma, d_i^\gamma$  with  $\gamma = T, F$ . Their size vectors are:

$$s(a_i^T) = (0, \epsilon N^i, r/64 + 2\tau + 1), \quad s(b_i^T) = (0, \epsilon N^i, r/64 + 4\tau + 1),$$

$$s(c_i^T) = (0, \epsilon N^i, r/64 + 8\tau + 1), \quad s(d_i^T) = (0, \epsilon N^i, r/64 + 16\tau + 1).$$

$$s(\tau_i^F) = s(\tau_i^T) + (0, 0, 1), \tau = a, b, c, d$$

- **Clause jobs.** For every clause  $e_j$ , if it contains two literals, then we construct two clause jobs,  $u_j^T$  and  $u_j^F$ . Otherwise it contains three literals, and we construct three clause jobs, one  $u_j^T$  and two  $u_j^F$ . Their size vectors are:

$$s(u_j^T) = (0, \epsilon N^{N+j}, r/4 + 2), \quad s(u_j^F) = (0, \epsilon N^{N+j}, r/4 + 4).$$

- **Dummy jobs.** We construct  $2n - m$  true dummy jobs  $\phi^T$  and  $m - n$  false dummy jobs  $\phi^F$ . Their size vectors are:

$$s(\phi^F) = (0, 0, r/16 + 4), \quad s(\phi^T) = (0, 0, r/16 + 2).$$

We finish the description of the element jobs and now define tuple jobs. Indeed, there is a one-to-one correspondence between tuple jobs and machines. For ease of description, we first construct machines, and then construct tuple jobs.

We construct  $8n$  machines, which are further divided into truth-assignment machines, clause machines and dummy machines. For a machine  $i$  we denote by  $g(i)$  its speed vector.

- **Truth-assignment machines.** For every variable  $z_i$ , we construct  $4n$  truth-assignment machines, denoted as  $(v_{i,1}, a_i, c_i), (v_{i,2}, b_i, d_i), (v_{i,3}, a_i, d_i), (v_{i,4}, b_i, c_i)$ . The symbol of a machine actually indicates the jobs that we will put on it. The speed vectors are:

$$g(v_{i,1}, a_i, c_i) = (N^{-4i-1}, N^{-i}, 1), \quad g(v_{i,2}, b_i, d_i) = (N^{-4i-2}, N^{-i}, 1),$$

$$g(v_{i,3}, a_i, d_i) = (N^{-4i-3}, N^{-i}, 1), \quad g(v_{i,4}, b_i, c_i) = (N^{-4i-4}, N^{-i}, 1).$$

- **Clause machines.** For every clause  $e_j$ : if the positive (or negative) literal  $z_i$  (or  $\neg z_i$ ) appears in it for the first time (i.e., it does not appear in  $e_k$  for  $k < j$ ), then we construct a clause machine  $(v_{i,1}, u_j)$  (or  $(v_{i,3}, u_j)$ ); if it appears for the second time, then we construct a clause machine  $(v_{i,2}, u_j)$  (or  $(v_{i,4}, u_j)$ ). The speed vectors are:

$$g(v_{i,k}, u_j) = (N^{-4i-k}, N^{-N-j}, 1).$$

- **Dummy machines.** Recall that for every variable, in all the clauses there are either one positive literal and two negative literals, or two positive literals and one negative literal. If  $z_i$  appears once and  $\neg z_i$  appears twice, then we construct a dummy machine  $(v_{i,2}, \phi)$ , otherwise we construct a dummy machine  $(v_{i,4}, \phi)$ . The speed vectors are:

$$g(v_{i,2}, \phi) = (N^{-4i-2}, 0, 1), \quad g(v_{i,4}, \phi) = (N^{-4i-4}, 0, 1).$$

According to our construction, it is not difficult to verify that if  $z_i$  appears once and  $\neg z_i$  appears twice, then we construct machines  $(v_{i,k}, u_{j_k})$  for  $k = 1, 3, 4, 1 \leq j_k \leq m$ , and machine  $(v_{i,2}, \phi)$ . Otherwise we construct machines  $(v_{i,k}, u_{j_k})$  for  $k = 1, 2, 3, 1 \leq j_k \leq m$ , and machine  $(v_{i,4}, \phi)$ . This completes the construction of machines.

- **Tuple jobs.** Finally, we construct tuple jobs. There is one tuple job corresponding to each machine. For simplicity, tuple jobs corresponding to truth-assignment, clause, dummy machines are called tuple-truth-assignment, tuple-clause, tuple-dummy jobs, respectively. We also use the symbol of a machine to denote its corresponding tuple job. The size vectors of tuple jobs are:

$$\begin{aligned}
s(v_{i,1}, a_i, c_i) &= (\epsilon N^{4i+1}, \epsilon N^i, 27r/32), & s(v_{i,2}, b_i, d_i) &= (\epsilon N^{4i+2}, \epsilon N^i, 27r/32), \\
s(v_{i,3}, a_i, d_i) &= (\epsilon N^{4i+3}, \epsilon N^i, 27r/32), & s(v_{i,4}, b_i, c_i) &= (\epsilon N^{4i+4}, \epsilon N^i, 27r/32). \\
s(v_{i,1}, u_j) &= (0, \epsilon N^{N+j}, 5r/8 + 10\tau), & s(v_{i,2}, u_j) &= (0, \epsilon N^{N+j}, 5r/8 + 20\tau), \\
s(v_{i,3}, u_j) &= (0, \epsilon N^{N+j}, 5r/8 + 18\tau), & s(v_{i,4}, u_j) &= (0, \epsilon N^{N+j}, 5r/8 + 12\tau). \\
s(v_{i,2}, \phi) &= (0, N^{2N}, 13r/16 + 20\tau), & s(v_{i,4}, \phi) &= (0, N^{2N}, 13r/16 + 12\tau).
\end{aligned}$$

Note that the size vectors of tuple-dummy jobs and tuple-clause jobs are actually independent of the index  $i$ .

This completes the construction of the whole scheduling instance. Recall that the processing time of a job on a machine is the inner product of the two corresponding vectors. Given our construction of machines and jobs, we have the following simple observation.

► **Observation 5.** *Let  $x$  be an arbitrary job whose size vector is  $s(x) = (s_1(x), s_2(x), s_3(x))$ . Then the processing time of  $x$  is at least  $s_3(x)$  on every machine. Furthermore, its processing time is  $s_3(x) + O(\epsilon)$  if one of the following holds:*

- $x$  is an element job and is scheduled on a machine whose symbol contains  $x$ ;
- $x$  is a tuple job and is scheduled on its corresponding machine.

We remark that, it is possible for a job  $x$  to have a processing time  $s_3(x) + O(\epsilon)$  on a machine even if the two conditions of the above observation do not hold, that is, the two conditions are not necessary.

The overall structure of our construction is similar to that of the paper [5] by Chen, Jansen and Zhang. We construct variable jobs corresponding to variables, clause jobs corresponding to clauses, truth-assignment jobs corresponding to the truth assignment of the SAT instance. Such kinds of jobs also appear in the reduction of [5] when they reduce 3SAT to the scheduling problem  $P||C_{max}$ . However, the reduction of Chen et al. [5] is for  $P||C_{max}$  which belongs to the rank 1 scheduling problem and does not work for higher ranks. To show APX-hardness, we need to construct completely different job processing times.

We first prove the following lemma.

► **Lemma 6.** *If there exists a truth assignment for  $I'_{sat}$  where every clause is satisfied by exactly one variable, then  $I_{sch}$  admits a feasible schedule whose makespan is  $r + O(\epsilon)$ .*

We give a brief overview of the proof and the reader may refer to the full version of this paper for details. It can be found at [https://www.researchgate.net/publication/313852592\\_Parameterized\\_and\\_approximation\\_results\\_for\\_scheduling\\_with\\_a\\_low\\_rank\\_processing\\_time\\_matrix](https://www.researchgate.net/publication/313852592_Parameterized_and_approximation_results_for_scheduling_with_a_low_rank_processing_time_matrix). We schedule jobs according to the first two columns of Table 1. Notice that the first two columns specify which job is on which machine, except that for an element job, say,  $a_i$ , it does not specify whether it is  $a_i^T$  or  $a_i^F$ . There are two possibilities regarding to the superscripts of element jobs on every machine, as is indicated by the third and fourth columns of Table 1. Either way ensures that the total processing times of jobs on each machine is  $r + O(\epsilon)$ . The technical part of the proof shows how to choose a proper way for every machine (based on the truth assignment of  $I'_{sat}$ ) so that all the jobs get scheduled.

■ **Table 1** Overview of the schedule.

machines	jobs	Feasible ways of scheduling	
$(v_{i,1}, a_i, c_i)$	$v_{i,1}, a_i, c_i, (v_{i,1}, a_i, c_i)$	$v_{i,1}^T, a_i^T, c_i^T$	$v_{i,1}^F, a_i^F, c_i^F$
$(v_{i,2}, b_i, d_i)$	$v_{i,2}, b_i, d_i, (v_{i,2}, b_i, d_i)$	$v_{i,2}^T, b_i^T, d_i^T$	$v_{i,2}^F, b_i^F, d_i^F$
$(v_{i,3}, a_i, d_i)$	$v_{i,3}, a_i, d_i, (v_{i,3}, a_i, d_i)$	$v_{i,3}^T, a_i^T, d_i^T$	$v_{i,3}^F, a_i^F, d_i^F$
$(v_{i,4}, b_i, c_i)$	$v_{i,4}, b_i, c_i, (v_{i,4}, b_i, c_i)$	$v_{i,4}^T, b_i^T, c_i^T$	$v_{i,4}^F, b_i^F, c_i^F$
$(v_{i,k}, u_j)$	$v_{i,k}, u_j, (v_{i,k}, u_j)$	$v_{i,k}^T, u_j^T$	$v_{i,k}^F, u_j^F$
$(v_{i,k}, \phi)$	$v_{i,k}, \phi, (v_{i,k}, \phi)$	$v_{i,k}^T, \phi^T$	$v_{i,k}^F, \phi^F$

► **Lemma 7.** *If there is a solution for  $I_{sch}$  whose makespan is strictly less than  $r + 1$ , then there exists a truth assignment for  $I'_{sat}$  where every clause is satisfied by exactly one literal.*

According to Observation 5, the total processing time of all jobs in any feasible solution is at least the summation of the third coordinates of all jobs, which is at least  $8nr$  with simple calculations. Let  $Sol^*$  be the solution whose makespan is strictly less than  $r + 1$ . We have the following structural lemma.

► **Lemma 8.** *In  $Sol^*$ , the followings are true:*

- *on a truth-assignment machine, there is exactly one tuple-truth-assignment job, two truth-assignment jobs and one variable job;*
- *on a clause machine, there is exactly one tuple-clause job, one clause job and one variable job;*
- *on a dummy machine, there is exactly one tuple-dummy job, one dummy job and one variable job.*

**Proof Idea.** The first and second coordinates of the speed and size vectors restrict the scheduling of jobs, e.g., by checking the second coordinate we can conclude that the processing time of a tuple-dummy job is  $\Omega(N)$  on any clause machine or truth-assignment machine, hence it has to be on a dummy machine. The third coordinate of a size vector gives a lower bound on the job processing time and allows us to derive some overall structure, e.g., each tuple job has a processing time at least  $5r/8$ , hence there can not be two tuple jobs on one machine. Given that the number of tuple jobs equals the number of machines, there is exactly one tuple job on one machine. Lemma 8 follows by combining the above basic idea with a careful analysis of job processing times. The reader may refer to the full version of this paper for all the details. ◀

A machine is called matched, if all the jobs on this machine coincide with the symbol of this machine, i.e., jobs are scheduled according to the second column of Table 1. Specifically, we say a machine is matched with respect to variable, or clause, or truth-assignment, or tuple jobs, if the variable, or clause, or truth-assignment, or tuple jobs on this machine coincide with the symbol of this machine.

► **Lemma 9.** *We may assume that every machine is matched with respect to variable jobs.*

**Proof.** Consider the eight jobs  $v_{n,k}^\gamma$  where  $\gamma = T, F, k = 1, 2, 3, 4$ . For any machine denoted as  $(v_{j,k}, *)$  or  $(v_{j,k}, *, *)$ , the first coordinate of its speed vector is  $N^{-4j-k}$ , thus the processing time of  $v_{n,k}$  on this machine becomes  $\Omega(\epsilon N)$  if  $j < n$ . Furthermore,  $v_{n,4}$  can only be on machines whose symbols are  $(v_{n,4}, *)$  or  $(v_{n,4}, *, *)$ , since if it is put on a machine whose symbol is  $(v_{n,k}, *)$  or  $(v_{n,k}, *, *)$  where  $k \in \{1, 2, 3\}$ , then its processing time also becomes  $\Omega(\epsilon N)$ . Notice that there are two jobs with the symbol  $v_{n,4}$  (one true job  $v_{n,4}^T$  and one



false job  $v_{n,4}^F$ ), and two machines with the symbol  $(v_{n,4}, *)$  or  $(v_{n,4}, *, *)$  (either machines  $(v_{n,4}, b_n, c_n)$  and  $(v_{n,4}, \phi)$ , or machines  $(v_{n,4}, b_n, c_n)$  and  $(v_{i,4}, u_{j_n})$  for some  $j_n$ ). According to Lemma 8, there is one variable job on every machine. Thus the two machines with the symbol  $(v_{n,4}, *)$  or  $(v_{n,4}, *, *)$  are matched with respect to variable jobs.

Next we consider the two variable jobs  $v_{n,3}$ . Using the same arguments as above, we can show that they can only be scheduled on a machine whose symbol is  $(v_{n,k}, *)$  or  $(v_{n,k}, *, *)$  where  $k \in \{3, 4\}$ . Furthermore, we have already shown that the variable job on a machine with the symbol  $(v_{n,4}, *)$  or  $(v_{n,4}, *, *)$  is  $v_{n,4}$ , and by Lemma 8 there can only be one variable job on every machine. Hence, the two jobs  $v_{n,3}$  can only be on the two machines whose symbols are  $(v_{n,3}, *)$  or  $(v_{n,3}, *, *)$ , and consequently these two machines are matched with respect to variable jobs.

Iteratively applying the above arguments we can prove that every machine is matched with respect to variable jobs.  $\blacktriangleleft$

We can further prove that every machine is matched with respect to clause jobs, tuple jobs and truth-assignment jobs, and therefore the following Lemma 10 is proved. The basic idea is similar to the proof of Lemma 9, but a more careful estimation of job processing times is required. A case by case analysis is needed several times to eliminate certain ways of scheduling. The reader may refer to the full version of this paper for details.

► **Lemma 10.** *We may assume that every machine is matched in  $Sol^*$ .*

Finally, we consider the superscripts of jobs on every machine. A machine is called truth benevolent, if except the tuple job, all the jobs on it are either all true or all false, i.e., jobs are scheduled according to the third or fourth column of Table 1. The following lemma follows by a case by case analysis showing that other ways of scheduling will lead to a total processing time larger than  $r + 1$  on some machine.

► **Lemma 11.** *Every machine is truth benevolent.*

**Proof of Lemma 7.** According to Lemma 11, for every  $1 \leq i \leq n$ , on truth-assignment machines jobs are either scheduled as  $(v_{i,1}^T, a_i^T, c_i^T)$ ,  $(v_{i,2}^T, b_i^T, d_i^T)$ ,  $(v_{i,3}^F, a_i^F, d_i^F)$ ,  $(v_{i,4}^F, a_i^F, c_i^F)$  or  $(v_{i,1}^F, a_i^F, c_i^F)$ ,  $(v_{i,2}^F, b_i^F, d_i^F)$ ,  $(v_{i,3}^T, a_i^T, d_i^T)$ ,  $(v_{i,4}^T, a_i^T, c_i^T)$ . If the former case happens, we let the variable  $z_i$  be false, otherwise we let  $z_i$  be true. We prove that, by assigning the truth value in this way, every clause of  $I'_{sat}$  is satisfied by exactly one literal.

Consider any clause, say,  $e_j$ . It contains two or three variables and we let them be  $v_{i_1, k_1}$ ,  $v_{i_2, k_2}$  and  $v_{i_3, k_3}$  where  $k_1, k_2, k_3 \in \{1, 2, 3, 4\}$  (if it contains two variables then  $v_{i_3, k_3}$  does not exist). Since there is one  $u_j^T$  and one or two  $u_j^F$ , we assume that  $u_j^T$  is scheduled with  $v_{i_1, k_1}^T$ .

We prove that  $e_j$  is satisfied by variable  $z_{i_1}$ . Notice that according to Lemma 10 and Lemma 11,  $u_j^T$  and  $v_{i_1, k_1}^T$  are scheduled together on machine  $(v_{i_1, k_1}, u_j)$ . There are two possibilities. Suppose  $k_1 \in \{1, 2\}$ . According to the construction of the scheduling instance, machine  $(v_{i_1, k_1}, u_j)$  is constructed if the positive literal  $z_i$  appears in clause  $e_j$  for the first or second time. According to our truth assignment in the paragraph above, variable  $z_i$  is true, for otherwise  $v_{i_1, k_1}^T$  is scheduled with  $a_i^T, c_i^T$  or  $b_i^T, d_i^T$ , thus  $e_j$  is satisfied by variable  $z_{i_1}$ . Otherwise  $k_1 \in \{3, 4\}$ . According to the construction of the scheduling instance, machine  $(v_{i_1, k_1}, u_j)$  is constructed if the negative literal  $\neg z_i$  appears in clause  $e_j$  for the first or second time. Again according to our truth assignment in the paragraph above, the variable  $z_i$  is false, thus  $e_j$  is satisfied by variable  $z_{i_1}$ .

We prove that  $e_j$  is *not* satisfied by variable  $z_{i_2}$  or  $z_{i_3}$ . Consider  $z_{i_2}$ . Notice that according to Lemma 10 and Lemma 11,  $u_j^F$  and  $v_{i_2, k_2}^F$  are scheduled together on machine  $(v_{i_2, k_2}, u_j)$ . There are two possibilities. Suppose  $k_2 \in \{1, 2\}$ . According to the construction of the

scheduling instance, machine  $(v_{i_2, k_2}, u_j)$  is constructed if the positive literal  $z_{i_2}$  appears in  $e_j$  for the first or second time. Meanwhile, variable  $z_{i_2}$  is false because otherwise  $v_{i_2, k_2}^F$  is scheduled with  $a_i^F, c_i^F$  or  $b_i^F, d_i^F$  according to our truth assignment of variables. Thus  $e_j$  is not satisfied by variable  $z_{i_2}$ . Similarly, we can prove that if  $k_2 \in \{3, 4\}$ ,  $e_j$  is not satisfied by variable  $z_{i_2}$ , either. The proof is the same for variable  $z_{i_3}$ , if it exists. ◀

## 4 Parameterized algorithms and lower bounds

### 4.1 Parameterizing by $p_{max}$ and $d$

We show  $R||C_{max}$  is FPT parameterized by  $p_{max}$  and the rank  $d$ . It is indeed a combination of a simple observation together with the following result by Knop and Koutecký [15].

► **Theorem 12** ([15]).  *$R||C_{max}$  is FPT parameterized by  $p_{max}$  and  $K$ , where  $K$  is the number of different kinds of machines.*

► **Remark.** In [15], machine kind is such defined that if two machines are of the same kind, then the processing time of every job is the same on them. Using our terminology,  $K$  is the number of distinct speed vectors. It is implicitly shown in [15] that the FPT algorithm runs in  $2^{O(\Theta^2 K \log p_{max})} + n^{O(1)}$  time, where  $\Theta$  is the number of distinct size vectors.

We observe that, if both the numbers of distinct speed vectors and size vectors are bounded by some function of  $p_{max}$  and  $d$ , then Theorem 2 follows directly from Theorem 12. In the following we show an even stronger result.

► **Lemma 13.** *Let  $\bar{p}$  be the number of distinct processing times in the matrix  $P = (p_{ij})_{m \times n}$ , and  $d$  be the rank of this matrix. There are at most  $\bar{p}^d + 1$  distinct speed vectors, and  $\bar{p}^d + 1$  distinct size vectors.*

**Proof.** We show that the number of distinct speed vectors is bounded by  $\bar{p}^d + 1$ . Due to symmetry the number of distinct size vectors is also bounded by the same value.

Consider all the size vectors. Since the matrix  $P$  has rank  $d$ , we are able to find  $d$  distinct size vectors that are linearly independent. Let them be  $v_1, v_2, \dots, v_d$ . Suppose there are  $n' \geq \bar{p}^d + 1$  distinct speed vectors and we consider each  $u_i \cdot v_1^T$  (recall that  $u_i$  is the speed vector of machine  $i$ ). As jobs have at most  $\bar{p}$  distinct processing times, the product  $u_i \cdot v_1^T$  can take at most  $\bar{p}$  distinct values. According to the pigeonhole principle there exist at least  $\lceil n' / \bar{p} \rceil \geq \bar{p}^{d-1} + 1$  distinct speed vectors leading to the same product. Similarly, among these speed vectors we can further select at least  $\bar{p}^{d-2} + 1$  ones such that their product with  $v_2$  are the same. Carry on the argument, eventually we can find at least 2 distinct speed vectors, say,  $u_1$  and  $u_2$ , such that their product with  $v_1, v_2, \dots, v_d$  are always the same, i.e.,  $(u_1 - u_2) \cdot v_i^T = 0$  for  $1 \leq i \leq d$ . However,  $v_1, v_2, \dots, v_d$  are linearly independent, hence  $u_1 - u_2 = 0$ , which contradicts the fact that  $u_1$  and  $u_2$  are different. ◀

Next we prove Theorem 3, which suggests that the FPT algorithm in Theorem 2 is essentially the best possible under ETH. We reduce from 3-dimensional matching.

### 3-Dimensional Matching (3DM)

**Input:** 3 disjoint sets of elements  $W = \{w_1, w_2, \dots, w_n\}$ ,  $X = \{x_1, x_2, \dots, x_n\}$ ,  $Y = \{y_1, y_2, \dots, y_n\}$  such that  $|W| = |X| = |Y| = n$ . A set  $T \subseteq W \times X \times Y$ .

**Output:** Decide whether there exists a perfect matching of size  $n$ , i.e., a subset  $T' \subseteq T$  such that  $|T'| = n$ , and for any two distinct triples  $(w, x, y), (w', x', y')$  it follows that  $w \neq w', x \neq x', y \neq y'$ .

The traditional NP-hardness proof (see, e.g., [8]) for the 3-dimensional matching problem reduces a 3SAT instance of  $n$  variables to a 3DM instance with  $O(n)$  elements, hence the following corollary follows.

► **Corollary 14.** *Assuming ETH, there is no  $2^{o(n)}$  time algorithm for 3DM.*

Given an arbitrary instance of 3DM, we construct in the following a scheduling instance with  $|T|$  machines and  $3|T|$  jobs such that the scheduling instance admits a feasible schedule of makespan at most  $11109\Gamma$  if and only if the 3DM instance admits a perfect matching, where  $\Gamma = \sum_{i=1}^{\tau} i \cdot (\tau - i)$  with integer  $\tau$  being the smallest integer such that  $\tau! \geq n$  (consequently,  $\tau = O(\log n / \log \log n)$ ). Furthermore, the scheduling instance we construct satisfies that  $d = O(\tau)$ ,  $p_{max} = \tau^{O(1)}$ . Now it is easy to verify that  $d \log p_{max} = O(\log n)$ . We claim that Theorem 3 follows from the reduction above. To see why, suppose on the contrary that Theorem 3 is false. Then there exists an algorithm of running time  $2^{2^{o(d \log p_{max})}}$  for  $R||C_{max}$ . We apply this algorithm to the constructed scheduling instance. As  $d \log p_{max} = O(\log n)$ , in  $2^{o(n)}$  time the algorithm determines whether the constructed scheduling instance admits a feasible schedule of makespan at most  $11109\Gamma$ , and consequently whether the given 3DM instance admits a perfect matching. This, however, is a contradiction to Corollary 14.

**Construction of the scheduling instance.** Note that  $\tau! \geq n$ , hence we can map each integer  $1 \leq i \leq n$  to a unique permutation of integers  $\{1, 2, \dots, \tau\}$ . Let  $\sigma$  be such a mapping. For ease of notation, we denote by  $\sigma_i$  the permutation that  $i$  is mapped to by  $\sigma$ . Consequently  $\sigma_i(k)$  denotes the integer on the  $k$ -th position of the permutation  $\sigma_i$ .

We construct  $|T|$  machines, each corresponding to one triple  $(w_i, x_j, y_k) \in T$ . The machine corresponding to  $(w_i, x_j, y_k)$  has the speed vector  $(1, \phi(w_i), \phi(x_j), \phi(y_k))$  where

$$\begin{aligned} \phi(w_i) &= (\sigma_i(1), \sigma_i(2), \dots, \sigma_i(\tau)), & \phi(x_j) &= (\sigma_j(1), \sigma_j(2), \dots, \sigma_j(\tau)), \\ \phi(y_k) &= (\sigma_k(1), \sigma_k(2), \dots, \sigma_k(\tau)). \end{aligned}$$

For every element  $z \in W \cup X \cup Y$ , let  $\eta(z)$  denote the number of occurrences of  $z$  in the set of triples  $T$ . We construct  $\eta(z)$  jobs for every element  $z$ . Among the  $\eta(z)$  jobs, there is one true job of size vector  $(g_T(z) \cdot \Gamma, \psi_w(z), \psi_x(z), \psi_y(z))$ . Each of the remaining  $\eta(z) - 1$  jobs is called a false job, having a size vector of  $(g_F(z) \cdot \Gamma, \psi_w(z), \psi_x(z), \psi_y(z))$ , where

$$\begin{aligned} \psi_w(w_i) &= (\tau - \sigma_i(1), \tau - \sigma_i(2), \dots, \tau - \sigma_i(\tau)), & \psi_w(x_j) &= \psi_w(y_k) = \underbrace{(0, 0, \dots, 0)}_{\tau}, \\ \psi_x(x_j) &= (\tau - \sigma_j(1), \tau - \sigma_j(2), \dots, \tau - \sigma_j(\tau)), & \psi_x(w_i) &= \psi_x(y_k) = \underbrace{(0, 0, \dots, 0)}_{\tau}, \\ \psi_y(y_k) &= (\tau - \sigma_k(1), \tau - \sigma_k(2), \dots, \tau - \sigma_k(\tau)), & \psi_y(w_i) &= \psi_y(x_j) = \underbrace{(0, 0, \dots, 0)}_{\tau}, \\ g_T(w_i) &= 10^2 + 4, & g_T(x_j) &= 10^3 + 1, & g_T(y_k) &= 10^4 + 1, \\ g_F(w_i) &= 10^2 + 2, & g_F(x_j) &= 10^3 + 2, & g_F(y_k) &= 10^4 + 2. \end{aligned}$$

We show that the constructed scheduling instance admits a feasible solution of makespan at most  $11109\Gamma$  if and only if the 3DM instance admits a perfect matching.

Suppose the given 3DM instance admits a perfect matching  $T'$ . For every  $(w_i, x_j, y_k) \in T'$ , we put the three true jobs corresponding to  $w_i, x_j, y_k$  onto the machine corresponding to this triple. It is easy to verify that the total processing time of the three jobs sum to exactly  $11109\Gamma$ . For every  $(w_{i'}, x_{j'}, y_{k'}) \in T \setminus T'$ , we put three false jobs corresponding to  $w_{i'}, x_{j'}, y_{k'}$  onto the machine corresponding to this triple. It is also easy to verify that the total

processing times sum up to  $11109\Gamma$ . Note that there is one true job corresponding to each element, while every element appears once in  $T'$ , all the jobs are scheduled and we derive a feasible schedule of makespan  $11109\Gamma$ .

Suppose the scheduling instance admits a feasible schedule of makespan bounded by  $11109\Gamma$ , we prove in the following that the *3DM* instance admits a perfect matching.

Consider the processing time of a job corresponding to  $z$  on a machine corresponding to  $(w_i, x_j, y_k)$ . The processing time is  $g_T(z) \cdot \Gamma + \lambda(z, (w_i, x_j, y_k))$ , if it is a true job, or  $g_F(z) \cdot \Gamma + \lambda(z, (w_i, x_j, y_k))$  otherwise. We observe that the processing time consists of two parts. The *machine-independent value*, which is  $g_T(z) \cdot \Gamma$  or  $g_F(z) \cdot \Gamma$  that only relies on the job, and the *machine-dependent value*, which is  $\lambda(z, (w_i, x_j, y_k))$ . The following lemma provides a lower bound on  $\lambda(z, (w_i, x_j, y_k))$ .

► **Lemma 15.** *For any element  $z$  and triple  $(w_i, x_j, y_k)$ , the following is true.*

$$\lambda(z, (w_i, x_j, y_k)) = (1, \phi(w_i), \phi(x_j), \phi(y_k)) \cdot (0, \psi_w(z), \psi_x(z), \psi_y(z))^T \geq \Gamma.$$

Furthermore, the equality holds if and only if  $z = w_i$  or  $z = x_j$  or  $z = y_k$ .

Lemma 15 follows immediately from the following *Rearrangement Inequality* [10].

► **Theorem 16 (Rearrangement Inequality).** *Let  $a_1 < a_2 < \dots < a_n$ ,  $b_1 < b_2 < \dots < b_n$  be two lists of real numbers, then*

$$a_n b_1 + a_{n-1} b_2 + \dots + a_1 b_n \leq a_{\pi(1)} b_1 + a_{\pi(2)} b_2 + \dots + a_{\pi(n)} b_n \leq a_1 b_1 + a_2 b_2 + \dots + a_n b_n$$

holds for any permutation  $\pi$ . Furthermore, the lower bound is attained if and only if  $\pi(i) = n + 1 - i$ , and the upper bound is attained if and only if  $\pi(i) = i$ .

► **Lemma 17.** *A job corresponding to an element  $z$  is scheduled on a machine corresponding to a triple that contains  $z$ .*

**Proof.** We sum up the processing time of all jobs. There are  $n$  true jobs and  $|T| - n$  false jobs corresponding to elements of  $W$ . The machine-independent value of these jobs sum up to  $104n\Gamma + 102(|T| - n)\Gamma = 102|T| \cdot \Gamma + 2n\Gamma$ . Similarly, it is easy to verify that the machine-independent value of jobs corresponding to elements of  $X$  and  $Y$  sum up to  $1001n\Gamma + 1002(|T| - n)\Gamma = 1002|T| \cdot \Gamma - n\Gamma$  and  $10001n\Gamma + 10002(|T| - n)\Gamma = 10002|T| \cdot \Gamma - n\Gamma$ , respectively. Hence the machine-independent value of all jobs sum up to  $11106|T| \cdot \Gamma$ . As the makespan is  $11109\Gamma$ , the total processing time of all jobs is at most  $11109|T| \cdot \Gamma$ , implying that the summation of machine-dependent value of all jobs is at most  $3|T| \cdot \Gamma$ . According to Lemma 15, the machine-dependent value of each job is at least  $\Gamma$ , regardless of which machine it is scheduled on. Given that there are  $3|T|$  jobs, the machine-dependent value of every job is exactly  $\Gamma$ . Again due to Lemma 15, a job corresponding to  $z$  must be scheduled on machine corresponding to a triple that contains  $z$ . ◀

For simplicity, we call a job corresponding to an element of  $W$  ( $X$  or  $Y$ ) as a  $w$ -job ( $x$ -job or  $y$ -job). We have the following lemma.

► **Lemma 18.** *There are three jobs on each machine, one  $w$ -job, one  $x$ -job and one  $y$ -job.*

**Proof.** Notice that the machine-dependent value of each job in the schedule is exactly  $\Gamma$ , hence the machine-independent value of jobs on each machine sum up to at most  $11106\Gamma$ . Notice that the machine-independent value of a  $y$ -job at least  $10^4\Gamma$ , there is at most one  $y$ -job on each machine. Furthermore, there are exactly  $|T|$  machines and  $y$ -jobs, hence, there is exactly one  $y$ -job on each machine. Similarly, we can show that there is one  $x$ -job and one  $w$ -job on each machine. ◀

Combining the above two lemmas, we have the following.

► **Lemma 19.** *On the machine corresponding to  $(w_i, x_j, y_k)$ , the three jobs correspond to  $w_i, x_j, y_k$ , respectively.*

Finally, we check whether jobs are true or false on each machine. Indeed, as the machine-independent value of the three jobs on each machine sum up to  $11106\Gamma$ , they are either all true jobs or all false jobs, hence, there are  $n$  machines on which all jobs are true jobs, and the triples corresponding to these machine form a perfect matching.

## 4.2 Parameterizing by $\bar{p}$ and $d$

We remark that, although it is not written explicitly, the general structural theorem in [9] actually implies an XP algorithm for  $R||C_{max}$  parameterized by  $\bar{p}$  and  $K$ , where  $K$  is the number of different kinds of machines. Combining this result with Lemma 13, Theorem 4 follows directly. For the completeness of this paper, we give all the proofs in the full version of this paper.

---

### References

- 1 Noga Alon, Yossi Azar, Gerhard J. Woeginger, and Tal Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998.
- 2 Aditya Bhaskara, Ravishankar Krishnaswamy, Kunal Talwar, and Udi Wieder. Minimum makespan scheduling with low rank processing times. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 937–947. Society for Industrial and Applied Mathematics, 2013.
- 3 Hans L. Bodlaender and Michael R. Fellows. W[2]-hardness of precedence constrained k-processor scheduling. *Operations Research Letters*, 18(2):93–97, 1995.
- 4 Vincenzo Bonifaci and Andreas Wiese. Scheduling unrelated machines of few different types. *arXiv preprint arXiv:1205.0974*, 2012.
- 5 Lin Chen, Klaus Jansen, and Guochuan Zhang. On optimality of exact and approximation algorithms for scheduling problems. Technical report, Christian-Albrechts-Universität Kiel, 2013. Report No. 1303. URL: [http://www.uni-kiel.de/journals/receive/jportal\\_jparticle\\_00000034](http://www.uni-kiel.de/journals/receive/jportal_jparticle_00000034).
- 6 Lin Chen, Deshi Ye, and Guochuan Zhang. An improved lower bound for rank four scheduling. *Operations Research Letters*, 42(5):348–350, 2014.
- 7 Lin Chen, Deshi Ye, and Guochuan Zhang. Online scheduling of mixed CPU-GPU jobs. *International Journal of Foundations of Computer Science*, 25(06):745–761, 2014.
- 8 Michael R. Garey and David S. Johnson. Computers and intractability: A guide to the theory of NP-completeness, 1979.
- 9 Michel X. Goemans and Thomas Rothvoß. Polynomiality for bin packing with a constant number of item types. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 830–839. Society for Industrial and Applied Mathematics, 2014.
- 10 Godfrey Harold Hardy, George Polya, and John Edensor Littlewood. *Inequalities*. Cambridge University Press, 1952.
- 11 Dorit S. Hochbaum and David B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM journal on computing*, 17(3):539–551, 1988.
- 12 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 653–662. IEEE, 1998.

- 13 Klaus Jansen. An EPTAS for scheduling jobs on uniform processors: using an milp relaxation with a constant number of integral variables. *SIAM Journal on Discrete Mathematics*, 24(2):457–485, 2010.
- 14 Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the gap for makespan scheduling via sparsification techniques. *arXiv preprint arXiv:1604.07153*, 2016.
- 15 Dušan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *arXiv preprint arXiv:1603.02611*, 2016.
- 16 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*, 46(1-3):259–271, 1990.
- 17 Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. *Mathematical Programming*, 154(1-2):533–562, 2015.
- 18 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226. ACM, 1978.
- 19 Jakob Schikowski. A PTAS for scheduling unrelated machines of few different types. In *SOFSEM 2016: Theory and Practice of Computer Science: 42nd International Conference on Current Trends in Theory and Practice of Computer Science, Harrachov, Czech Republic, January 23-28, 2016, Proceedings*, volume 9587, page 290. Springer, 2016.
- 20 Evgeny V. Shchepin and Nodari Vakhania. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, 33(2):127–133, 2005.
- 21 Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.
- 22 René van Bevern, Robert Brederick, Laurent Bulteau, Christian Komusiewicz, Nimrod Talmon, and Gerhard J. Woeginger. Precedence-constrained scheduling problems parameterized by partial order width. *arXiv preprint arXiv:1605.00901*, 2016.

# Fractional Coverings, Greedy Coverings, and Rectifier Networks<sup>\*†</sup>

Dmitry Chistikov<sup>1</sup>, Szabolcs Iván<sup>2</sup>, Anna Lubiw<sup>3</sup>, and Jeffrey Shallit<sup>4</sup>

- 1 Department of Computer Science, University of Oxford, Oxford, UK  
chdir@cs.ox.ac.uk
- 2 Department of Foundations of Computer Science, University of Szeged, Szeged, Hungary  
szabivan@inf.u-szeged.hu
- 3 School of Computer Science, University of Waterloo, Waterloo, Canada  
alubiw@cs.uwaterloo.ca
- 4 School of Computer Science, University of Waterloo, Waterloo, Canada  
shallit@cs.uwaterloo.ca

---

## Abstract

A rectifier network is a directed acyclic graph with distinguished sources and sinks; it is said to compute a Boolean matrix  $M$  that has a 1 in the entry  $(i, j)$  iff there is a path from the  $j$ th source to the  $i$ th sink. The smallest number of edges in a rectifier network that computes  $M$  is a classic complexity measure on matrices, which has been studied for more than half a century.

We explore two techniques that have hitherto found little to no applications in this theory. They build upon a basic fact that depth-2 rectifier networks are essentially weighted coverings of Boolean matrices with rectangles. Using *fractional* and *greedy* coverings (defined in the standard way), we obtain new results in this area.

First, we show that all *fractional* coverings of the so-called full triangular matrix have cost at least  $n \log n$ . This provides (a fortiori) a new proof of the tight lower bound on its depth-2 complexity (the exact value has been known since 1965, but previous proofs are based on different arguments). Second, we show that the *greedy* heuristic is instrumental in tightening the upper bound on the depth-2 complexity of the Kneser-Sierpiński (disjointness) matrix. The previous upper bound is  $O(n^{1.28})$ , and we improve it to  $O(n^{1.17})$ , while the best known lower bound is  $\Omega(n^{1.16})$ . Third, using *fractional* coverings, we obtain a form of direct product theorem that gives a lower bound on unbounded-depth complexity of Kronecker (tensor) products of matrices. In this case, the *greedy* heuristic shows (by an argument due to Lovász) that our result is only a logarithmic factor away from the “full” direct product theorem. Our second and third results constitute progress on open problem 7.3 and resolve, up to a logarithmic factor, open problem 7.5 from a recent book by Jukna and Sergeev (in Foundations and Trends in Theoretical Computer Science (2013)).

**1998 ACM Subject Classification** G.2.1 Combinatorics

**Keywords and phrases** rectifier network, OR-circuit, biclique covering, fractional covering, greedy covering

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.23

---

\* A full version of the paper is available at <http://arxiv.org/abs/1509.07588>.

† This research was done while Dmitry Chistikov was a postdoctoral researcher at the Max Planck Institute for Software Systems in Kaiserslautern and Saarbrücken, Germany, and was supported by the ERC Synergy award ImPACT. He is currently supported by the ERC grant AVS-ISS (648701). Szabolcs Iván was supported by the NKFI grant no. 108448.



## 1 Introduction

Introduced in the 1950s, *rectifier networks* are one of the oldest and most basic models in the theory of computing. They are directed acyclic graphs with distinguished input and output nodes; a rectifier network is said to *compute* (or *express*) the Boolean matrix  $M$  that has a 1 in the entry  $(i, j)$  iff there is a path from the  $j$ th input to the  $i$ th output. Equivalently, rectifier networks can be viewed as Boolean circuits that consist entirely of OR gates of arbitrary fan-in. This simple model of computation has attracted a lot of attention [11], because it captures the “topological” core of other models: complexity bounds for rectifier networks extend in one way or another to Boolean circuits (i.e., circuits with Boolean gates) and to switching circuits [24, 21].

Given a matrix  $M$ , what is the smallest number of edges in a rectifier network that computes  $M$ ? Denote this number by  $\text{OR}(M)$  – this is a complexity measure on Boolean matrices. This measure is fairly well understood: it is known, by the results of Nechiporuk [23], that the maximum of  $\text{OR}(M)$  grows as  $n^2/2 \log n$  as  $n \rightarrow \infty$  if  $M$  is  $n \times n$ ; it is also known that random  $n \times n$ -matrices have complexity very close to  $n^2/2 \log n$ . The “shape” of these two facts is reminiscent of the standard circuit complexity of Boolean functions over AND, OR, and NOT gates – but for them, the maximum is  $2^n/n$  instead of  $n^2/2 \log n$ .

However, much more is, in fact, known about the measure  $\text{OR}(\cdot)$ : there are explicit sequences of matrices that have complexity within a factor  $n^{o(1)}$  from the maximum. In this context, such factors are usually regarded as small (note that, in contrast, for circuits over AND, OR, and NOT gates, exhibiting a single sequence of functions that require a superlinear number of gates would be a tremendous breakthrough). In fact, nowadays a range of methods are available for obtaining upper and lower bounds on  $\text{OR}(M)$  for specific matrices  $M$ ; we refer the interested reader to the recent book by Jukna and Sergeev [11].

Many natural questions, however, remain open. Jukna and Sergeev list 19 open problems about  $\text{OR}(\cdot)$  and related complexity measures. Several of them refer to very restricted submodels, such as rectifier networks of depth 2: that is, networks where all paths contain (at most) 2 edges. A depth-2 rectifier network expressing a matrix  $M$  is essentially a *covering* of  $M$  – a collection of (rectangular) all-1 submatrices of  $M$  whose disjunction is  $M$ . In our work, we look into the corresponding complexity measure  $\text{OR}_2(\cdot)$  as well as  $\text{OR}(\cdot)$ . We build upon the connection between rectifier networks and (weighted) set coverings and explore two ideas that have previously found few applications in the study of rectifier networks: they are associated with fractional and greedy coverings respectively.

*Fractional coverings* are a generalization of usual set coverings. In the usual set cover problem, each set  $S$  can be either included or not included in the solution (i.e., in the covering); in the fractional version each set can be partially included: a solution assigns to each set  $S$  a real number  $x_S \in [0; 1]$ , and for every element  $s$  of the universe the sum  $\sum_{s \in S} x_S$  should be equal to or exceed 1. In other words, fractional coverings arise from linear relaxation of the integer program that expresses the set cover problem. *Greedy coverings* are, in contrast, usual coverings; they are the outcome of applying the standard greedy heuristic to an instance of the set cover problem: at each step, the algorithm picks a set  $S$  that covers the largest number of yet uncovered elements  $s$ . In our work, we use fractional and greedy coverings to obtain estimates on the values of  $\text{OR}_2(M)$  and  $\text{OR}(M)$ .

### Our results

First, we demonstrate that  $\text{OR}_2(T_n) = n(\lfloor \log_2 n \rfloor + 2) - 2^{\lfloor \log_2 n \rfloor + 1}$ , where  $T_n$  is the so-called full triangular matrix: an upper-triangular matrix that has 1s everywhere above the main



diagonal and 0s on the diagonal and below. In this problem, the upper bound is easy and the challenge is to prove the lower bound. This bound was obtained by Krichevskii [15], and our paper provides a new proof of independent interest (which also serves as an illustration of our techniques). In fact, we prove an even stronger statement: all *fractional* coverings of  $T_n$  have large associated cost (Theorem 4). To this end, we take the linear program that expresses the fractional set cover problem and find a good feasible solution to the dual program. The value of this solution then gives a lower bound on the cost of all feasible solutions to the primal – that is, on the cost of fractional coverings. Since integral coverings are just a special case of fractional coverings, the result follows.

Second, we improve the upper bound on the value of  $\text{OR}_2(D_n)$ , where  $D_n$  is the disjointness matrix, also known as the Kneser-Sierpiński matrix. This constitutes progress on open problem 7.3 in Jukna and Sergeev’s book [11], where the previously known bounds are obtained. The previous upper bound is  $O(n^{1.28})$ , and our Theorem 9 improves it to  $O(n^{1.17})$ , while the best known lower bound is  $\Omega(n^{1.16})$ . To achieve this improvement, we subdivide the instance of the weighted set cover problem (in which the optimal value is  $\text{OR}_2(D_n)$ ) into  $\text{polylog}(n)$  natural subproblems and reduce them, by imposing an additional restriction, to instances of unweighted set cover problems. We then solve these instances with the *greedy* heuristic; the upper bound in the analysis invokes the so-called greedy covering lemma by Sapozhenko [27], also known as the Lovász–Stein theorem [17, 31]. This gives us the desired upper bound on  $\text{OR}_2(D_n)$ ; in fact, the greedy strategy turns out to be optimal, and the optimal exponent in  $\text{OR}_2(D_n)$  comes from a numerical optimization problem. As an intermediate result we determine, up to a polylogarithmic factor, the value of  $\text{OR}_2(D_k^m)$  where  $D_k^m$  is the adjacency matrix of the Kneser graph on  $2\binom{k}{m}$  vertices.

Finally, we obtain (Theorem 13) a form of direct product theorem for the  $\text{OR}(\cdot)$  measure:  $\text{OR}(K \otimes M) \geq \text{rk}_v^*(K) \cdot \text{OR}(M)$ , where  $K \otimes M$  denotes the Kronecker product of matrices  $K$  and  $M$ , and  $\text{rk}_v^*(K)$  is a fractional analogue of the Boolean rank of  $K$ . This resolves, up to a logarithmic factor, open problem 7.5 in the list of Jukna and Sergeev [11], which asks for the lower bound of  $\text{rk}_v(K) \cdot \text{OR}(M)$  where  $\text{rk}_v(K) \geq \text{rk}_v^*(K)$  is the Boolean rank of  $K$ . (In fact, a related question for unambiguous rectifier networks, or SUM-circuits, is originally due to Find et al. [5]; our technique applies to this model as well, giving an analogous inequality for the measure  $\text{SUM}(\cdot)$ , see Corollary 14.) Suppose  $K$  is an  $m \times n$  matrix; then, by the argument due to Lovász [18], the *greedy* heuristic shows that  $\text{rk}_v^*(K) \geq \text{rk}_v(K)/(1 + \log mn)$ , so our lower bound is indeed at most a logarithmic factor away from the “full” direct product theorem. To prove our lower bound, we take the linear programming formulation of the *fractional* set cover problem for the matrix  $K$  and use components of the optimal solution to the dual program to guide our argument. It is interesting to see how reasoning about coverings, or, equivalently, about depth-2 rectifier networks, enables us to obtain meaningful lower bounds on the size of rectifier networks that have unbounded depth.

## 2 Discussion and related work

We use the matrix language in this paper, but all results can be restated in terms of biclique coverings of bipartite graphs.

The  $\text{OR}_2$ -complexity of full triangular matrices,  $T_n$ , is tightly related to results on biclique coverings of complete undirected (non-bipartite) graphs from the early days of the theory of computing. The  $n \log n$  lower bound, in one form or another, was known

to Hansel [7], Krichevskii [15], Katona and Szemerédi [14], and Tarján [32].<sup>1</sup> On the one hand, our lower bound is obtained in a setting with an asymmetry restriction: for  $\text{OR}_2(T_n)$ , one needs to cover entries  $(i, j)$  with  $i < j$  in the matrix, whereas in biclique coverings of undirected graphs, it suffices to cover either of  $(i, j)$  and  $(j, i)$ . On the other hand, to the best of our knowledge, ours is the only proof that goes via linear programming (LP) duality and provides a tight lower bound on the size of *fractional* coverings. This result is new; moreover, we are not aware of any other lower bounds for rectifier networks that come from feasible solutions to the LP dual (in approximation algorithms, a related technique is known as “dual fitting” [37, Section 9.4]). Apart from purely combinatorial considerations, the interest in the problem is motivated by its applications in formula and switching-circuit complexity of the Boolean threshold-2 function (which takes on the value 1 iff at least two of its inputs are set to 1). For more context, see treatments by Radhakrishnan [26] and Lozhkin [20].

As for the **greedy heuristics**, while we are not the first to use them in the context of rectifier networks (see Nechiporuk [24, Sect. 1.6]), they were never previously used to study the complexity of individual matrices. The disjointness matrix,  $D_n$ , which we apply this technique to, is a well-studied object in communication complexity [16]; it is a discrete version of the Sierpiński triangle. For arbitrary-depth networks, the values  $\text{OR}(D_n)$  and  $\text{SUM}(D_n)$  are  $\Theta(n \log n)$ , as shown by Boyar and Find [1] and Selezneva [28].<sup>2</sup> In depth 2, the previous bounds are due to Jukna and Sergeev [11]; it is unknown if greedy heuristics are also of use for SUM-circuits, as our upper bound for  $D_n$  does not extend to this model (our coverings are not partitions).

**Direct sum and direct product theorems** in the theory of computing are statements of the following form: when faced with several instances of the same problem on different independent inputs, there is no better strategy than solving each instance independently.<sup>3</sup> For rectifier networks, these questions are associated with the complexity of Kronecker (tensor) products of matrices. Indeed, denote the  $k \times k$ -identity matrix by  $I_k$ , then  $I_k \otimes M$  is the block-diagonal matrix with  $k$  copies of  $M$  on the diagonal. It is not difficult to show that  $\text{OR}(I_k \otimes M) \geq k \cdot \text{OR}(M)$ , and a natural generalization asks whether  $\text{OR}(K \otimes M) \geq \text{rk}_V(K) \cdot \text{OR}(M)$  for any matrix  $K$  – see Find et al. [5] and Jukna and Sergeev [11, Sections 2.4, 3.6, and open problem 7.5]. To date, this inequality is only known to hold in special cases. For example, Find et al. [5] can show this lower bound when the matrix  $K$  has a fooling set of size  $\text{rk}_V(K)$ ; however, the size of the largest fooling set does not approximate the Boolean rank, as observed, e.g., by Gruber and Holzer [6] (they use the graph-theoretic language, with bipartite dimension instead of  $\text{rk}_V$ ). As another example, denote by  $|M|$  the number of 1s in the matrix  $M$  and assume that  $M$  has no all-1 submatrices of size  $(k+1) \times (l+1)$ . Then the inequality  $\text{OR}(M) \geq |M|/kl$  is a well-known lower bound due to Nechiporuk [24], subsequently rediscovered by Mehlhorn [21], Pippenger [25], and Wegener [36]; Jukna and Sergeev [11, Theorem 3.20] extend it to  $\text{OR}(K \otimes M) \geq \text{rk}_V(K) \cdot |M|/kl$  for any square matrix  $K$ . To the best of our knowledge, the current literature has no stronger lower bounds on the OR-complexity of Kronecker products; our Theorem 13 answers this need, coming logarithmically close to the bound in question. For SUM-complexity, the previous state of the art and our contribution are analogous to the OR case. The related notion of a fractional biclique cover has appeared, e.g., in the papers of Watts [35] and Jukna and Kulikov [10].

<sup>1</sup> Not all of these arguments compute the *exact* value of  $\text{OR}_2(T_n)$ .

<sup>2</sup> Recall that the  $\text{SUM}(\cdot)$  measure corresponds to *unambiguous* rectifier networks, in which every input-output pair is connected by at most one path; or, equivalently, to arithmetic circuits over nonnegative integers with addition (SUM) gates. For any matrix  $M$ ,  $\text{OR}(M) \leq \text{SUM}(M)$  and  $\text{OR}_2(M) \leq \text{SUM}_2(M)$ .

<sup>3</sup> In some contexts, the terms “direct sum theorem” and “direct product theorem” have slightly different meanings [29], but in the current context we do not distinguish between them.

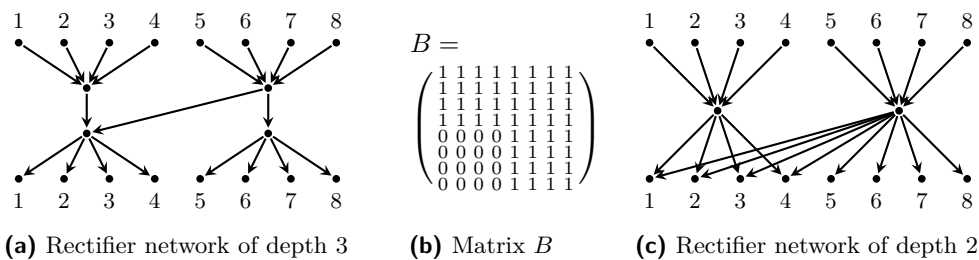


Figure 1 Illustrations for Example 1.

Also related to our work is the study of the size of smallest biclique coverings, under the name of the bipartite dimension of a graph (as opposed to the cost of such coverings and the  $\text{OR}_2$ -complexity; see Section 3). This quantity corresponds to the Boolean rank of a matrix and is known to be PSPACE-hard to compute [6] and NP-hard to approximate to within a factor of  $n^{1-\epsilon}$  [2]. Finally, we note that results on  $\text{OR}_2$ -complexity have corollaries for **descriptive complexity** of regular languages. Indeed, take a language where all words have length two,  $L \subseteq \Sigma \cdot \Delta$ , with  $\Sigma = \{a_1, \dots, a_m\}$  and  $\Delta = \{a_1, \dots, a_n\}$ . Let  $M^L$  be its characteristic  $m \times n$  matrix:  $M^L_{i,j} = 1$  iff  $a_i \cdot a_j \in L$ . Then  $\text{OR}_2(M^L)$  coincides with the alphabetic length of the shortest regular expression for  $L$ ; for example, it follows from Corollary 5 that the optimal regular expression for the language  $L_n = \{a_i a_j \mid 1 \leq i < j \leq n\}$  has  $n(\lfloor \log_2 n \rfloor + 2) - 2^{\lfloor \log_2 n \rfloor + 1}$  occurrences of letters ( $\Sigma = \Delta = \{a_1, \dots, a_n\}$ ). The values of  $\text{OR}(M^L)$  and  $\text{OR}_2(M^L)$  are also related to the size of the smallest nondeterministic finite automata accepting  $L$ ; see [8] and the full version of the present paper (see <http://arxiv.org/abs/1509.07588>) for details.

### 3 Rectifier networks and coverings

#### Rectifier networks

Define a *rectifier network* with  $n$  inputs and  $m$  outputs as a 4-tuple  $\mathcal{N} = (V, E, \text{in}, \text{out})$ , where  $V$  is a set of vertices,  $E \subseteq V^2$  a set of edges such that the directed graph  $G_{\mathcal{N}} = (V, E)$  is acyclic, and  $\text{in}: \{1, \dots, n\} \rightarrow V$  and  $\text{out}: \{1, \dots, m\} \rightarrow V$  are injective functions whose images contain only sources (resp., only sinks) of  $G_{\mathcal{N}}$ . The network  $\mathcal{N}$  is said to have *size*  $|E|$ .

A rectifier network  $\mathcal{N}$  *expresses* a Boolean  $m \times n$  matrix  $M = M(\mathcal{N})$  such that  $M_{ij} = 1$  if  $G_{\mathcal{N}}$  contains a directed path from  $\text{in}(j)$  to  $\text{out}(i)$  and  $M_{ij} = 0$  otherwise. A rectifier network  $\mathcal{N}$  is said to have *depth*  $d$  if all maximal paths in  $G_{\mathcal{N}}$  have exactly  $d$  edges. Given a Boolean matrix  $A \in \{0, 1\}^{m \times n}$ , let  $\text{OR}_2(A)$  denote the smallest size of a depth-2 rectifier network that expresses  $A$  and let  $\text{OR}(A)$  denote the smallest size of any rectifier network that expresses  $A$ .

This notation is justified by the following observation. A rectifier network  $\mathcal{N}$  may be viewed as a circuit: its Boolean inputs are located at the vertices  $\text{in}(\{1, \dots, n\})$ , and gates at all other vertices compute the disjunction (Boolean OR) of their inputs. From this point of view, the circuit computes a linear operator over the monoid  $(\{0, 1\}, \text{OR})$ , and the matrix of this linear operator is exactly the Boolean matrix expressed by the rectifier network  $\mathcal{N}$ .

► **Example 1.** A depth-3 rectifier network is shown in Figure 1a. It expresses the matrix  $B$  in Figure 1b, showing that  $\text{OR}_3(B) \leq 19$ . In fact, this network is optimal and  $\text{OR}_3(B) = 19$ ; see the full version of the paper for a proof of a more general statement. At the same time,  $\text{OR}_2(B) = 20$ : the upper bound is achieved by the network in Figure 1c, and the lower bound is due to Jukna and Sergeev [11, Theorem 3.18].

$$\begin{array}{lll}
\sum_{S \in \mathcal{F}} w(S) x_S \rightarrow \min & \sum_{S \in \mathcal{F}} w(S) x_S \rightarrow \min & \sum_{u \in U} y_u \rightarrow \max \\
x_S \in \{0, 1\} \text{ for all } S \in \mathcal{F} & 0 \leq x_S \leq 1 \text{ for all } S \in \mathcal{F} & y_u \geq 0 \text{ for all } u \in U \\
\sum_{\substack{S \in \mathcal{F}: \\ u \in S}} x_S \geq 1 \text{ for all } u \in U & \sum_{\substack{S \in \mathcal{F}: \\ u \in S}} x_S \geq 1 \text{ for all } u \in U & \sum_{u \in S} y_u \leq w(S) \text{ for all } S \in \mathcal{F}
\end{array}$$

(a) Integer program                      (b) Linear relaxation                      (c) Dual of the linear relaxation

■ **Figure 2** Integer and linear programs for the set cover problem.

### Coverings of Boolean matrices

Let us describe an alternative way of defining the function  $\text{OR}_2(\cdot)$ . Given a Boolean matrix  $A \in \{0, 1\}^{m \times n}$ , a *rectangle* (or a 1-rectangle) is a pair  $(R, C)$ , where  $R \subseteq \{1, \dots, m\}$  and  $C \subseteq \{1, \dots, n\}$ , such that for all  $(i, j) \in R \times C$  we have  $A_{ij} = 1$ . A rectangle  $(R, C)$  is said to *cover* all pairs  $(i, j) \in R \times C$ . The *cost* of a rectangle  $(R, C)$  is defined as  $|R| + |C|$ .

Suppose a matrix  $A$  is fixed; then a collection of rectangles is called a *covering* of  $A$  if for every  $(i, j) \in \{1, \dots, m\} \times \{1, \dots, n\}$  there exists a rectangle in the collection that covers  $(i, j)$ . The *cost* of a collection is the sum of costs of all its rectangles.

Given a Boolean matrix  $A \in \{0, 1\}^{m \times n}$ , the *cost* of  $A$  is defined as the smallest cost of a covering of  $A$ . It is not difficult to show that the cost of  $A$  equals  $\text{OR}_2(A)$  as defined above.

Similarly, we can think of minimizing the *size* of a covering, i.e., the number of rectangles in a collection instead of their total cost. The smallest size of a covering of  $A$  is called the *OR-rank* (or the *Boolean rank*) of  $A$ , denoted  $\text{rk}_\vee A$ .

## 4 Fractional and greedy coverings

In the rest of the paper we interpret the covering problems for Boolean matrices as special cases of the general set cover problem. In this section we recall this general setting and present two main techniques that we apply: linear programming duality and greedy heuristics.

An instance of the (*weighted*) *set cover* problem consists of a set  $U$ , a family of its subsets,  $\mathcal{F} \subseteq 2^U$ , and a weight function, which is a mapping  $w: \mathcal{F} \rightarrow \mathbb{N}$ . Every set  $S \in \mathcal{F}$  is said to *cover* all elements  $s \in S \subseteq U$ . The goal is to find a subfamily  $\mathcal{F}' \subseteq \mathcal{F}$  that is a *covering* (i.e., it covers all elements from  $U$ :  $\bigcup_{S \in \mathcal{F}'} S = U$ ) and has the smallest possible total weight (i.e., it minimizes the functional  $\sum_{S \in \mathcal{F}'} w(S)$  amongst all coverings). In the *unweighted* version of the problem,  $w(S) = 1$  for all  $S \in \mathcal{F}$ , so the total weight of a covering is just its *size* (number of elements in  $\mathcal{F}'$ ). In both versions,  $\mathcal{F}$  is usually assumed to be a *feasible* solution, which means that every  $s \in U$  belongs to at least one set from  $\mathcal{F}$ : that is,  $\bigcup_{S \in \mathcal{F}} S = U$ .

It is instructive, throughout this section, to have particular instances of the set cover problem in mind, namely those of covering Boolean matrices with rectangles as in Section 3. In the following sections, we refer to them as *weighted* and *unweighted set covering formulations*; their optimal solutions correspond to the values of  $\text{OR}_2(A)$  and  $\text{rk}_\vee A$  respectively.

### Fractional coverings

The set cover problem can easily be recast as an integer program: see Figure 2a. For each  $S \in \mathcal{F}$ , this program has an integer variable  $x_S \in \{0, 1\}$ : the interpretation is that  $x_S = 1$  if and only if  $S \in \mathcal{F}'$ , and the constraints require that every element is covered. *Feasible* solutions are in a natural one-to-one correspondence with coverings of  $U$ , and the optimal value in the program is the smallest weight of a covering.

The *linear programming relaxation* of this integer program is obtained by interpreting variables  $x_S$  over reals: see Figure 2b. Now  $0 \leq x_S \leq 1$  for each  $S \in \mathcal{F}$ . Feasible solutions to this program are called *fractional coverings*. Suppose the optimal cost in the original set cover problem is  $\tau$ . Then the integer program in Figure 2a has optimal value  $\tau$ , and its relaxation in Figure 2b optimal value  $\tau^* \leq \tau$ .

Finally, define the *dual* of this linear program: this is also a linear program, and it has a (real) variable  $y_u$  for each element  $u \in U$ ; see Figure 2c. This is a maximization problem, and its optimal value coincides with  $\tau^*$  by the strong duality theorem.

The following lemma (see, e.g., [12]) summarizes the properties needed for the sequel.

► **Lemma 2.** *If  $(y_u)_{u \in U}$  is a feasible solution to the dual, then  $\sum_{u \in U} y_u \leq \tau^* \leq \tau$ . There exists a feasible solution to the dual,  $(y_u^*)_{u \in U}$ , such that  $\sum_{u \in U} y_u^* = \tau^*$ .*

We use the first part of Lemma 2 in Section 5 to obtain a lower bound on  $\tau$  and the second part in Section 7 to associate “weights” with 1-elements in the matrix.

### Greedy coverings

The greedy heuristic for the unweighted set cover problem works as follows. It maintains the set of uncovered elements, initially  $U$ , and iteratively adds to  $\mathcal{F}'$  (which is initially empty) a set  $S \in \mathcal{F}$  which covers the largest number of yet-uncovered elements. Any covering obtained by this (nondeterministic) procedure is called a *greedy covering*. (There is a natural extension to the weighted version as well.)

A standard analysis of the greedy heuristic is performed in the framework of approximation algorithms: the size of a greedy covering is at most  $O(\log |U|)$  times larger than that of the optimal covering [3, 18]. But for our purposes a different upper bound will be more convenient: an “absolute” upper bound in terms of the “density” of the instance. Such a bound is given by the following result, which is substantially less well-known:

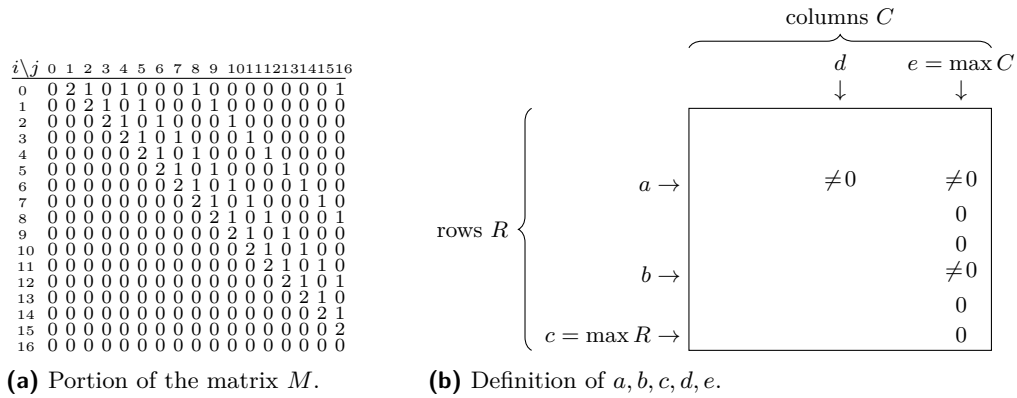
► **Lemma 3 (greedy covering lemma).** *Suppose every element  $s \in U$  is contained in at least  $\gamma|\mathcal{F}|$  sets from  $\mathcal{F}$ , where  $0 < \gamma \leq 1$ . Then the size of any greedy covering does not exceed*

$$\left\lceil \frac{1}{\gamma} \ln^+(\gamma|U|) \right\rceil + \frac{1}{\gamma},$$

where  $\ln^+(x) = \max(0, \ln x)$  and  $\ln x$  is the natural logarithm.

Several versions of the lemma can be found in the literature. It was proved for the first time in 1972 by Sapozhenko [27] and appears in later textbooks [33, Lemma 9 in Section 3, pp. 136–137], [34, pp. 134–135]. A slightly different form, attributed to Stein [31] and Lovász [17], was independently obtained later and is sometimes known as the Lovász–Stein theorem; yet another proof is due to Karpinski and Zelikovsky [13]. Recent treatments with applications and more detailed discussion can be found in Deng et al. [4] and in Jukna’s textbook [9, pp. 34–37].

Since the upper bound of Lemma 3 is hardly a standard tool in theoretical computer science as of now, a remark on the proof is in order. A standalone proof goes via the following fact: on each step of the greedy algorithm the number of yet-uncovered elements shrinks by a constant factor, determined by the density parameter  $\gamma$  and the size of the instance. Alternatively, one can use the result due to Lovász [17] that the size of any greedy covering is within a factor of  $1 + \log |U|$  from the optimal *fractional* covering. Since assigning the value  $(\min_{s \in U} |\{S \in \mathcal{F} : s \in S\}|)^{-1} = 1/\gamma|U|$  to all  $x_S$ ,  $S \in \mathcal{F}$ , in the linear program in Figure 2b leads to a feasible solution, an upper bound of  $(1/\gamma) \cdot (1 + \log |U|)$  follows.



■ **Figure 3** Illustrations for the proof of Theorem 4.

We use Lemma 3 in Section 6 to obtain an upper bound on the  $\text{OR}_2$ -complexity of Kneseer-Sierpiński matrices. We remark that instead of greedy coverings one can use random coverings to essentially the same effect (cf. Deng et al. [4]).

### 5 Lower bound for the full triangular matrices

Define the  $n \times n$  full triangular matrix  $T_n = (t_{ij})_{0 \leq i, j < n}$  by  $t_{ij} = 1$  if  $i < j$  and  $t_{ij} = 0$  otherwise. This matrix  $T_n$  is the adjacency matrix of the Hasse diagram of the strict linear order  $0 < 1 < \dots < n - 1$ ; it has 1s everywhere above the main diagonal and 0s on the diagonal and below. In this section, we study the smallest size of depth-2 rectifier networks that express  $T_n$ .

Define  $s(n) = n(\lceil \log_2 n \rceil + 2) - 2^{\lceil \log_2 n \rceil + 1}$  for  $n \geq 1$ . Note that  $s(n)$  is the so-called binary entropy function, sequence [A003314](#) in Sloane's *Encyclopedia of Integer Sequences* [30]. Its properties were studied previously by Morris [22] because of its connection with mergesort.

► **Theorem 4.** All fractional coverings of  $T_n$  have cost of at least  $s(n)$ .

► **Corollary 5.**  $\text{OR}_2(T_n) = s(n)$ .

We discuss the proof of Theorem 4 below. Note that the equality of Corollary 5 gives the exact value of  $\text{OR}_2(T_n)$ . The upper bound of the theorem is an easy divide-and-conquer argument, and the main challenge is to obtain the lower bound.

Consider the weighted set covering formulation for  $T_n$ , where the optimal value is  $\text{OR}_2(T_n)$  as discussed in Section 4. By Lemma 2, it suffices to find a feasible solution to the dual linear program with the value  $s(n)$ . Our feasible solution is given by a certain infinite diagonal matrix  $M$ , with rows and columns indexed by the natural numbers, defined as follows:

$$M_{i,j} = \begin{cases} 2, & \text{if } j - i = 1; \\ 1, & \text{if } j - i = 2^q \text{ for some } q \geq 1; \\ 0, & \text{otherwise.} \end{cases}$$

The first 17 rows and columns of  $M$  are displayed in Figure 3a. Notice that each row is a shift, by 1, of the preceding row.

► **Lemma 6.** The sum of the elements of  $M^{(n)}$ , the  $n \times n$  upper left submatrix of  $M$ , is equal to  $s(n)$ .

**Proof.**  $M^{(n+1)}$  is obtained from  $M^{(n)}$  by concatenating a row of 0's on the bottom, and a column that contains a single 2 and 1's corresponding to the powers of 2 that are  $\leq n$ . In other words,  $s(n+1) = s(n) + \lfloor \log_2 n \rfloor + 2$ . The result now follows by an easy induction. ◀

► **Lemma 7.**  $y_{i,j} = M_{i,j}$  for  $0 \leq i < j < n$  is a feasible solution to the dual program.

**Proof.** Let  $R$  correspond to a choice of rows of  $M$  and  $C$  to a choice of columns. To prove feasibility, we need to see that for each pair of nonempty sets  $R, C \subseteq \{0, 1, \dots, n-1\}$  with  $\max R < \min C$  – only such pairs  $(R, C)$  are rectangles of  $T_n$ , because the matrix has 0s everywhere on the main diagonal and below – we have

$$\sum_{\substack{i \in R \\ j \in C}} M_{i,j} \leq |R| + |C|. \quad (1)$$

Suppose there exists a counterexample to (1). Among all counterexamples to (1), consider one with the smallest possible value of  $|R| + |C|$ . If  $|R| = 1$  then since at most one entry in each row is 2 and all others are either 0 or 1, we clearly have  $\sum_{\substack{i \in R \\ j \in C}} M_{i,j} \leq |C| + 1 = |R| + |C|$ . Hence  $|R| \geq 2$ . The same argument applies if  $|C| = 1$ . Thus the minimal counterexample to (1) has at least two rows and columns.

We now observe that the row sum of each row in our counterexample is at least 2. For if it is 0 or 1 we could omit that row, and (1) would still be violated. The same argument applies to the column sums. We can prove the following statement (the details can be found in the full version of the paper):

► **Claim 8.** *Suppose there are at least two nonzero elements in the submatrix of  $M$  formed by rows  $0, 1, \dots, b$  and column  $e$  of  $M$ . Then  $e \leq 2b$ .*

Now let us assume that our minimal counterexample has  $c = \max R$ . Let  $e = \max C$ . Since column  $e$  has 2 nonzero elements, by the Claim above we know  $e \leq 2c$ . Now let  $b$  be the largest element  $\leq c$  in  $R$  for which there is a nonzero element in column  $e$ ; this must exist since column  $e$  has at least two nonzero elements. Let  $a$  be any row  $< b$  in  $R$  with a nonzero element in column  $e$ . Again, this must exist since column  $e$  has at least two nonzero elements. Finally, let  $d$  be any column  $< e$  in  $C$  with a nonzero element in row  $a$ . This must exist because every row in  $R$  has at least two nonzero elements. We claim  $d \leq c$ .

To see this, note that  $b = e - 2^j \leq c$  for some  $j \geq 0$ . (In fact,  $j = \lceil \log_2(e - c) \rceil$ .) Then we must have  $a = e - 2^k \geq 0$  where  $k \geq j + 1$ . Then  $d - a = 2^\ell$  for some  $\ell$ . So  $d - a = d - (e - 2^k) = 2^\ell$  and hence  $d = e + 2^\ell - 2^k$ . Since  $d < e$ , we have  $\ell < k$ . So  $d \leq e + 2^{k-1} - 2^k = e - 2^{k-1} \leq e - 2^j = b \leq c$ . This is illustrated in Figure 3b.

Now  $\max R < \min C$ , but  $d \leq c$  while  $d \in C$  and  $c \in R$ , a contradiction. Hence there are no minimal counterexamples and no counterexamples at all. Thus (1) holds. It follows that  $M$  represents a feasible solution. This concludes the proof of Lemma 7. ◀

Let us complete the proof of Theorem 4. Apply the first part of Lemma 2 to the weighted set covering formulation of the problem and take the solution  $y_{i,j} = M_{i,j}$ ,  $0 \leq i < j < n$ , as described above. This solution has value  $s(n)$  by Lemma 6 and is feasible by Lemma 7. Hence, all fractional coverings have cost at least  $s(n)$ .

## 6 Upper bound for Kneser-Sierpiński matrices

Suppose  $n = 2^k$ . A *Kneser-Sierpiński matrix* (or a *disjointness matrix*) of size  $2^k \times 2^k$  is the matrix  $D_n$  defined as follows. Rows and columns of the matrix are indexed from 0 to  $2^k - 1$ .

The matrix has a 1 at all positions  $(i, j)$  such that  $i$  and  $j$  have no common 1 in their binary expansion; all other elements of the matrix are 0.

Note that if we identify each number from  $\{0, \dots, n-1\}$  with a subset of  $\{1, \dots, k\}$  in the natural way, then  $D_n$  is naturally associated with a Boolean function that maps a pair of subsets of  $\{1, \dots, k\}$  to 1 if they are disjoint, and to 0 if they have an element in common. An alternative way to define  $D_n$  is by a recurrence  $D_{2n} = \begin{pmatrix} D_n & D_n \\ D_n & 0^n \end{pmatrix}$  for  $n \geq 1$ ;  $D_1 = (1)$ ; here subsets of  $\{1, \dots, k\}$  are ordered lexicographically. Using the antilexicographic order for rows and the lexicographic order for columns would lead to a lower triangular matrix.

What is the size of smallest depth-2 rectifier networks that express Kneser-Sierpiński matrices? Jukna and Sergeev [11, Lemma 4.2] prove that

$$n^{\frac{1}{2} \log 5} / \text{polylog}(n) \leq \text{OR}_2(D_n) \leq n^{\log(1+\sqrt{2})} \cdot \text{polylog}(n), \quad (2)$$

and in this section, we prove the following result:

► **Theorem 9.**  $\text{OR}_2(D_n) \leq n^{\log(9/4)} \cdot \text{polylog}(n)$ .

Note that  $\frac{1}{2} \log 5 \approx 1.16096$ ,  $\log(9/4) \approx 1.16993$ , and  $\log(1 + \sqrt{2}) \approx 1.27$ .

Suppose  $n = 2^k$  as above, and let  $D_{[k]}^{x,y}$  be the submatrix of  $D_n$  whose rows and columns correspond to  $x$ -sized and  $y$ -sized subsets of  $\{1, \dots, k\}$ , respectively. This matrix  $D_{[k]}^{x,y}$  has size  $\binom{k}{x} \times \binom{k}{y}$ . If  $x = y$ , then  $D_{[k]}^{x,x}$  is the adjacency matrix of the Kneser graph [19].

For  $0 \leq y \leq x \leq k$ , write  $z = (k - x - y)/2$  and  $f(x, y) = \binom{k}{x, z, k-x-z} / \binom{2z}{z}$ .<sup>4</sup> Jukna and Sergeev [11, Lemma 4.2] show that all coverings of  $D_{[k]}^{x,x}$  have cost at least  $f(x, x) / \text{poly}(k)$ , and this gives the lower bound in equation (2): taking  $x = 0.4k$  brings  $f(x, x)$  to its maximum of  $n^{\frac{1}{2} \log 5}$ , if we disregard factors polylogarithmic in  $n = 2^k$ . Our Theorem 9 follows from Lemmas 10 and 12 below.

► **Lemma 10.** *There exists a covering of  $D_{[k]}^{x,y}$  with cost at most  $f(x, y) \cdot \text{poly}(k)$ .*

**Proof.** Consider  $\mathcal{F}$ , the family of all *ordered bipartitions* of  $\{1, \dots, k\}$  into sets of size  $x + z$  and  $y + z$ , where  $z = (k - x - y)/2$ . Technically, an ordered bipartition is simply a subset of  $\{1, \dots, k\}$ , but it is more instructive to view it as an ordered pair: this subset and its complement. Every such bipartition,  $(S, \bar{S})$ , corresponds to a (maximal) rectangle in  $D_{[k]}^{x,y}$ ; elements of  $D_{[k]}^{x,y}$  covered by the rectangle are pairs  $(X, Y)$  of disjoint sets that *respect* the bipartition:  $X \subseteq S$  and  $Y \subseteq \bar{S}$ .

Use the greedy covering lemma (Lemma 3) for the unweighted set covering formulation with  $\mathcal{F}$ . There are  $\binom{k}{x+z}$  bipartitions in this family, and every pair of disjoint sets  $(X, Y)$  of size  $x$  and  $y$  respects  $\binom{2z}{z}$  of them, so  $\gamma = \binom{2z}{z} / \binom{k}{x+z}$  and any greedy covering will contain at most  $N$  sets, where

$$N = \frac{\binom{k}{x+z}}{\binom{2z}{z}} \cdot (1 + \ln(4^k)) + 1 = \frac{\binom{k}{x+z}}{\binom{2z}{z}} \cdot \text{poly}(k).$$

For every bipartition in the covering, the corresponding 1-rectangle in  $D_{[k]}^{x,y}$  will include  $\binom{x+z}{z}$  rows and  $\binom{y+z}{z}$  columns; its cost will be at most  $2 \binom{x+z}{z}$  as  $y \leq x$ . So the total cost of the covering will not exceed

$$\binom{x+z}{z} \cdot 2N = \frac{2 \binom{k}{x+z} \binom{x+z}{z} \cdot \text{poly}(k)}{\binom{2z}{z}} = \frac{\binom{k}{x, z, k-x-z} \cdot \text{poly}(k)}{\binom{2z}{z}} = f(x, y) \cdot \text{poly}(k). \quad \blacktriangleleft$$

<sup>4</sup> We use the standard notation for multinomial coefficients:  $\binom{k}{a, b, c} = \frac{k!}{a! b! c!}$  provided that  $a + b + c = k$ .



► **Corollary 11.** *Suppose  $0 \leq m \leq k/2$  and let  $D_k^m = D_{[k]}^{m,m}$  be the adjacency matrix of the (bipartite) Kneser graph: vertices in each part are size- $m$  subsets of  $\{1, \dots, k\}$ , and two vertices from different parts are adjacent if and only if the subsets are disjoint. Then  $d(m, k)/\text{poly}(k) \leq \text{OR}_2(D_k^m) \leq d(m, k) \cdot \text{poly}(k)$  where  $d(m, k) = \binom{k}{m, k/2-m, k/2} / \binom{k-2m}{k/2-m}$ .*

► **Lemma 12.** *If  $0 \leq y \leq x \leq k$ , then  $f(x, y) \leq 2^{k \log(9/4)} \cdot \text{poly}(k)$ , and there exists a pair  $(x^*, y^*)$  such that  $f(x^*, y^*) \geq 2^{k \log(9/4)} / \text{poly}(k)$ .*

To complete the proof of Theorem 9, it remains to note that a union of coverings of matrices  $D_{[k]}^{x,y}$  for all pairs  $x, y$  with  $0 \leq x, y \leq k$  constitutes a covering of  $D_n$ . For  $0 \leq y \leq x \leq k$ , the coverings are constructed by Lemma 10, and for  $x \leq y$  the construction just swaps the roles of  $x$  and  $y$ . Since there are only  $(k+1)^2 = \text{polylog}(n)$  pairs  $x, y$  in total, the desired upper bound follows from Lemma 12.

► **Remark.** Although Theorem 9 leaves a gap between the bounds on  $\text{OR}_2(D_n)$ , the greedy strategy is, in fact, optimal up to a  $\text{polylog}(n)$  factor: For each  $D_{[k]}^{x,y}$ , it suffices to use bipartitions into sets of size  $\ell$  and  $k - \ell$ , for some  $\ell = \ell(k; x, y)$ . (See the full version of the paper for details.) Our choice of  $\ell$  in Lemma 10 is  $\ell = x + (k - x - y)/2$ , and the optimal choice,  $\ell = \ell^*(k; x, y)$ , will deliver a tight upper bound on  $\text{OR}_2(D_n)$ . Numerical experiments seem to indicate that the actual value of  $\text{OR}_2(D_n)$  is within a  $\text{polylog}(n)$  factor from  $n^{\frac{1}{2} \log 5}$ , but no formal proof is known to us.

## 7 Lower bound for Kronecker products

Given two matrices  $K \in \{0, 1\}^{m_1 \times n_1}$  and  $M \in \{0, 1\}^{m_2 \times n_2}$ , their *Kronecker* (or *tensor*) *product* is the Boolean matrix  $K \otimes M$  of size  $(m_1 \cdot m_2) \times (n_1 \cdot n_2)$  defined as follows. Its rows are indexed by pairs  $(i_1, i_2)$  and its columns by pairs  $(j_1, j_2)$  where  $1 \leq i_s \leq m_s$  and  $1 \leq j_s \leq n_s$  for  $s = 1, 2$ . The entry of  $K \otimes M$  at position  $((i_1, i_2), (j_1, j_2))$  is defined as  $K_{i_1, j_1} \cdot M_{i_2, j_2}$ .

In this section we prove a lower bound on the  $\text{OR}(\cdot)$ -measure of Kronecker products. Recall that the Boolean rank  $\text{rk}_V(K)$  is the optimal value of the unweighted set covering formulation (as in Figure 2a) where the set of 1-entries in the matrix  $K$  is covered by all-1 rectangles. In the linear relaxation of this problem (as in Figure 2b), the goal is to assign weights  $w(R) \in [0, 1]$  to each 1-rectangle  $R$  such that  $\sum_{(i,j) \in R} w(R) \geq 1$  for each 1-entry  $(i, j)$  of  $K$ , minimizing  $\sum w(R)$ . Let the *fractional rank*  $\text{rk}_V^*(K)$  be the optimal value of this linear relaxation. The integrality gap result for the set cover problem [17] and the duality theorem imply that  $\text{rk}_V(K)/(1 + \log m_1 n_1) \leq \text{rk}_V^*(K) \leq \text{rk}_V(K)$ . In the graph-theoretic language, the number  $\text{rk}_V^*(K)$  is the *fractional biclique cover number*, denoted by  $bc^*(G)$  where  $K$  is the adjacency matrix of the (bipartite) graph  $G$ . Fractional rank is known to be bounded from below by the fooling set number, see Watts [35, Theorem 2.2].

► **Theorem 13.** *For any pair  $K, M$  of Boolean matrices,  $\text{OR}(K \otimes M) \geq \text{rk}_V^*(K) \cdot \text{OR}(M)$ .*

**Proof.** First consider the unweighted set covering formulation for  $K$ , where the optimal value is  $\text{rk}_V(K)$  as discussed in Section 4, and take its linear relaxation, with the optimal value  $\text{rk}_V^*(K)$ . By Lemma 2, there is an assignment of weights to 1-elements of this matrix,  $w(i, j) \in [0, 1]$  for all  $(i, j)$  with  $K_{i,j} = 1$ , such that the following two conditions are satisfied (see Figure 2c). First, for each 1-rectangle  $R \times C$  of  $K$ , the sum  $\sum_{(i,j) \in R \times C} w(i, j)$  is at most 1. Second,  $\sum_{(i,j): K_{i,j}=1} w(i, j) = \text{rk}_V^*(K)$ .

Now let  $\mathcal{N} = (V, E, \text{in}, \text{out})$  be a rectifier network of size  $\text{OR}(K \otimes M)$  that expresses  $Q = K \otimes M$ , where  $K$  and  $M$  have size as above. For an edge  $e \in E$ , let  $\text{To}(e) \subseteq$

$\{1, \dots, m_1\} \times \{1, \dots, m_2\}$  be the set of row indices  $(i_1, i_2)$  of  $Q$  such that the node  $\text{out}(i_1, i_2)$  is reachable from the target of  $e$ . Similarly, let  $\text{From}(e) \subseteq \{1, \dots, n_1\} \times \{1, \dots, n_2\}$  be the set of column indices  $(j_1, j_2)$  of  $Q$  such that the source of  $e$  is reachable from  $\text{in}(j_1, j_2)$ . Then  $R(e) = (\text{To}(e), \text{From}(e))$  is a rectangle of  $Q$ . Moreover, define  $\pi_s((i_1, i_2), (j_1, j_2)) = (i_s, j_s)$  for  $s = 1, 2$  and  $\pi_s(R) = \{\pi_s(r, c) : (r, c) \in R\}$ . Then  $\pi_1(R(e))$  and  $\pi_2(R(e))$  are rectangles in  $K$  and  $M$  respectively.

We assign real weights based on  $w$  to each edge  $e$  of  $\mathcal{N}$  by the following rule:

$$w'(e) = \sum_{(i,j) \in \pi_1(R(e))} w(i, j).$$

Since  $\pi_1(R(e))$  is a rectangle of  $K$ , one of the constraints on  $w$  ensures that  $w'(e) \leq 1$  for each edge  $e$  of  $\mathcal{N}$ . Consequently,  $\sum_{e \in E} w'(e) \leq |E| = \text{OR}(K \otimes M)$ ; furthermore, the following chain of inequalities holds:

$$\begin{aligned} \text{OR}(K \otimes M) &\geq \sum_{e \in E} w'(e) = \sum_{e \in E} \sum_{(i_1, j_1) \in \pi_1(R(e))} w(i_1, j_1) \\ &= \sum_{(i_1, j_1) : K_{i_1, j_1} = 1} w(i_1, j_1) \cdot |\{e \in E : (i_1, j_1) \in \pi_1(R(e))\}| \\ &= \sum_{(i_1, j_1) : K_{i_1, j_1} = 1} w(i_1, j_1) \cdot |\{e \in E : i_1 \in \pi_1(\text{To}(e)), j_1 \in \pi_1(\text{From}(e))\}|. \end{aligned} \quad (3)$$

Fix an arbitrary entry  $(i_1, j_1)$  of  $K$  with  $K_{i_1, j_1} = 1$ . Consider the subgraph  $\mathcal{N}_{j_1 \rightsquigarrow i_1}$  of  $\mathcal{N}$  induced by the nodes that are reachable from some source of the form  $\text{in}(j_1, j_2)$  and from which a node of the form  $\text{out}(i_1, i_2)$  is reachable – in other words, take all nodes and edges on all paths from  $\text{in}(j_1, j_2)$  to  $\text{out}(i_1, i_2)$  for some  $i_2, j_2$ . Then, since  $K_{i_1, j_1} = 1$ , the node  $\text{out}(i_1, i_2)$  is reachable from  $\text{in}(j_1, j_2)$  in  $\mathcal{N}_{j_1 \rightsquigarrow i_1}$  if and only if  $M_{i_2, j_2} = 1$ . So the network  $\mathcal{N}_{j_1 \rightsquigarrow i_1}$  expresses  $M$  (with the mappings  $\text{in}'(j_2) = \text{in}(j_1, j_2)$  and  $\text{out}'(i_2) = \text{out}(i_1, i_2)$ ). Hence, the number of edges in  $\mathcal{N}_{j_1 \rightsquigarrow i_1}$  is at least  $\text{OR}(M)$ . But by our definitions, the relations  $i_1 \in \pi_1(\text{To}(e))$  and  $j_1 \in \pi_1(\text{From}(e))$  hold together exactly for the edges  $e$  of  $\mathcal{N}$  present in  $\mathcal{N}_{j_1 \rightsquigarrow i_1}$ . Thus  $|\{e \in E : i_1 \in \pi_1(\text{To}(e)), j_1 \in \pi_1(\text{From}(e))\}| \geq \text{OR}(M)$  and we conclude from equation (3) that

$$\text{OR}(K \otimes M) \geq \sum_{(i_1, j_1) : K_{i_1, j_1} = 1} w(i_1, j_1) \cdot \text{OR}(M) = \text{rk}_V^*(K) \cdot \text{OR}(M). \quad \blacktriangleleft$$

**► Remark.** Let  $\text{SUM}(K)$  be the smallest size of an *unambiguous* rectifier network that expresses  $K$ . A rectifier network is unambiguous if for all  $i, j$  it has at most one path from  $\text{in}(j)$  to  $\text{out}(i)$ . Such networks are also known under the names of SUM-circuits [11] and cancellation-free circuits [1]. The same construction as above also proves the inequality  $\text{SUM}(K \otimes M) \geq \text{rk}_V^*(K) \cdot \text{SUM}(M)$ .

**► Corollary 14.** For any pair of matrices  $K \in \{0, 1\}^{m_1 \times n_1}$  and  $M \in \{0, 1\}^{m_2 \times n_2}$ , and  $L \in \{\text{OR}, \text{SUM}\}$ , it holds that  $L(K \otimes M) \geq \text{rk}_V(K) \cdot L(M) / (1 + \log m_1 n_1)$ .

**Acknowledgements.** We are grateful to Stasys Jukna, Alexander Kulikov, Igor Sergeev, and anonymous reviewers for comments and discussions.

---

## References

- 1 Joan Boyar and Magnus Gausdal Find. Cancellation-free circuits in unbounded and bounded depth. *Theor. Comput. Sci.*, 590:17–26, 2015. doi:10.1016/j.tcs.2014.10.014.

- 2 Parinya Chalermsook, Sandy Heydrich, Eugenia Holm, and Andreas Karrenbauer. Nearly tight approximability results for minimum biclique cover and partition. In *Algorithms – ESA 2014 – 22th Annual European Symposium, Wrocław, Poland, September 8–10, 2014. Proceedings*, pages 235–246, 2014. doi:10.1007/978-3-662-44777-2\_20.
- 3 V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.
- 4 Dameng Deng, P. C. Li, G. H. J. van Rees, and Yuan Zhang. The Stein-Lovasz theorem and its applications to some combinatorial arrays. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 77:17–31, 2011.
- 5 Magnus Find, Mika Göös, Matti Järvisalo, Petteri Kaski, Mikko Koivisto, and Janne H. Korhonen. Separating OR, SUM, and XOR circuits. *Journal of Computer and System Sciences*, 82(5):793–801, 2016.
- 6 Hermann Gruber and Markus Holzer. Finding lower bounds for nondeterministic state complexity is hard. *Electronic Colloquium on Computational Complexity (ECCC)*, 13(027), 2006. URL: <http://eccc.hpi-web.de/eccc-reports/2006/TR06-027/index.html>.
- 7 G. Hansel. Nombre minimal de contacts de fermeture nécessaires pour réaliser une fonction booléenne symétrique de  $n$  variables. *C. R. Acad. Sc. Paris*, 258(25):6037–6040, 1964. In French.
- 8 Szabolcs Iván, Ádám Dániel Lelkes, Judit Nagy-György, Balázs Szörényi, and György Turán. Biclique coverings, rectifier networks and the cost of  $\epsilon$ -removal. In *Descriptive Complexity of Formal Systems – 16th International Workshop, DCFS 2014, Turku, Finland, August 5–8, 2014. Proceedings*, pages 174–185, 2014. doi:10.1007/978-3-319-09704-6\_16.
- 9 Stasys Jukna. *Extremal Combinatorics*. Springer-Verlag, 2011. 2nd edition.
- 10 Stasys Jukna and Alexander S. Kulikov. On covering graphs by complete bipartite subgraphs. *Discrete Mathematics*, 309(10):3399–3403, 2009. doi:10.1016/j.disc.2008.09.036.
- 11 Stasys Jukna and Igor Sergeev. Complexity of linear Boolean operators. *Foundations and Trends in Theoretical Computer Science*, 9(1):1–123, 2013. Available at <http://lovelace.thi.informatik.uni-frankfurt.de/~jukna/Knizka/linear.pdf>. doi:10.1561/04000000063.
- 12 Howard Karloff. *Linear Programming*. Birkhäuser, 2008. 2nd printing.
- 13 Marek Karpinski and Alexander Zelikovsky. Approximating dense cases of covering problems. In *Network design: connectivity and facilities location*, volume 40 of *DIMACS*, pages 169–178. AMS, 1998.
- 14 Gyula Katona and Endre Szemerédi. On a problem of graph theory. *Studia Scientiarum Mathematicarum Hungarica*, 2:23–28, 1967.
- 15 R. E. Krichevskii. A minimal monotone contact scheme for a Boolean function of  $n$  variables. In *Diskretnyj Analiz (Discrete Analysis)*, volume 5, pages 89–92. Institute for Mathematics in the Siberian Section of the Academy of Sciences, Novosibirsk, 1965. In Russian.
- 16 Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.
- 17 László Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13(4):383–390, 1975. doi:10.1016/0012-365X(75)90058-8.
- 18 László Lovász. A kombinatorika minimax tételeiről. *Matematikai Lapok*, 26:209–264, 1976. In Hungarian.
- 19 László Lovász. Kneser’s conjecture, chromatic number, and homotopy. *J. Comb. Theory, Ser. A*, 25(3):319–324, 1978. doi:10.1016/0097-3165(78)90022-5.

- 20 S. A. Lozhkin. On minimal  $\pi$ -circuits for monotone symmetric functions with threshold 2. *Diskretnaya Matematika*, 17(4):108–110, 2005. In Russian. English translation in *Discrete Mathematics and Applications* 15(5) (2005), 475–477.
- 21 Kurt Mehlhorn. Some remarks on Boolean sums. *Acta Inf.*, 12:371–375, 1979. doi:10.1007/BF00268321.
- 22 Robert Morris. Some theorems on sorting. *SIAM J. Appl. Math.*, 17:1–6, 1969.
- 23 E. I. Nechiporuk. Rectifier networks. *Soviet Physics Doklady*, 8:5–7, March 1963.
- 24 E. I. Nechiporuk. On the topological principles of self-correction. *Problemy Kibernetiki*, 21:5–102, 1969. In Russian. English translation in: *Systems Theory Res.* 21 (1970), 1–99.
- 25 Nicholas Pippenger. On another Boolean matrix. *Theor. Comput. Sci.*, 11:49–56, 1980. doi:10.1016/0304-3975(80)90034-1.
- 26 Jaikumar Radhakrishnan. Entropy and counting. In J. C. Misra, editor, *Computational Mathematics, Modelling and Algorithms*. Narosa Publishers, New Delhi, 2003. Available online at <http://www.tcs.tifr.res.in/~jaikumar/Papers/EntropyAndCounting.pdf>.
- 27 Alexander Sapozhenko. On the complexity of disjunctive normal forms obtained with a gradient algorithm. In *Diskretnyj Analiz (Discrete Analysis)*, volume 21, pages 62–71. Institute for Mathematics in the Siberian Section of the Academy of Sciences, Novosibirsk, 1972. In Russian.
- 28 S. N. Selezneva. Lower bound on the complexity of finding polynomials of Boolean functions in the class of circuits with separated variables. *Computational Mathematics and Modeling*, 24(1):146–152, 2013. doi:10.1007/s10598-013-9166-1.
- 29 Alexander A. Sherstov. Strong direct product theorems for quantum communication and query complexity. *SIAM J. Comput.*, 41(5):1122–1165, 2012. doi:10.1137/110842661.
- 30 N. J. A. Sloane. On-line encyclopedia of integer sequences. Electronic resource at <http://oeis.org>.
- 31 S. K. Stein. Two combinatorial covering theorems. *J. Comb. Theory, Ser. A*, 16(3):391–397, 1974. doi:10.1016/0097-3165(74)90062-4.
- 32 T. G. Tarján. Complexity of lattice-configurations. *Studia Scientiarum Mathematicarum Hungarica*, 10:203–211, 1975.
- 33 Yu. L. Vasilyev and V. V. Glagolev. Metrical properties of disjunctive normal forms. In S. V. Yablonsky and O. B. Lupanov, editors, *Discrete mathematics and mathematical questions of cybernetics*, pages 99–148. Nauka, Moscow, 1974. In Russian.
- 34 J. L. Wassiljew and W. W. Glagolew. Metrische Eigenschaften alternativer Normalformen. In S. W. Jablonski and O. B. Lupanow, editors, *Diskrete Mathematik und Mathematische Fragen der Kybernetik*, volume 71 of *Mathematische Reihe*, pages 100–144. Birkhäuser Basel, 1980. Translation of [33]. In German. doi:10.1007/978-3-0348-5543-3\_3.
- 35 Valerie L. Watts. Fractional biclique covers and partitions of graphs. *Electr. J. Comb.*, 13(1), 2006. URL: [http://www.combinatorics.org/Volume\\_13/Abstracts/v13i1r74.html](http://www.combinatorics.org/Volume_13/Abstracts/v13i1r74.html).
- 36 Ingo Wegener. A new lower bound on the monotone network complexity of Boolean sums. *Acta Inf.*, 13:109–114, 1980. doi:10.1007/BF00263988.
- 37 David P. Williamson and David B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011.

# Separability of Reachability Sets of Vector Addition Systems\*

Lorenzo Clemente<sup>1</sup>, Wojciech Czerwiński<sup>2</sup>, Sławomir Lasota<sup>3</sup>, and Charles Paperman<sup>4</sup>

- 1 University of Warsaw, Warsaw, Poland
- 2 University of Warsaw, Warsaw, Poland
- 3 University of Warsaw, Warsaw, Poland
- 4 University of Tübingen, Tübingen, Germany

---

## Abstract

Given two families of sets  $\mathcal{F}$  and  $\mathcal{G}$ , the  $\mathcal{F}$ -separability problem for  $\mathcal{G}$  asks whether for two given sets  $U, V \in \mathcal{G}$  there exists a set  $S \in \mathcal{F}$ , such that  $U$  is included in  $S$  and  $V$  is disjoint with  $S$ . We consider two families of sets  $\mathcal{F}$ : modular sets  $S \subseteq \mathbb{N}^d$ , defined as unions of equivalence classes modulo some natural number  $n \in \mathbb{N}$ , and unary sets, which extend modular sets by requiring equality below a threshold  $n$ , and equivalence modulo  $n$  above  $n$ . Our main result is decidability of modular and unary separability for the class  $\mathcal{G}$  of reachability sets of Vector Addition Systems, Petri Nets, Vector Addition Systems with States, and for sections thereof.

**1998 ACM Subject Classification** D.2.2 [Design Tools and Techniques] Petri Nets, F.1.1 [Theory of Computation] Models of Computation, F.2.2 [Nonnumerical Algorithms and Problems] Computations on Discrete Structures, F.3.1 [Specifying and Verifying and Reasoning about Programs] Mechanical Verification

**Keywords and phrases** separability, Petri nets, modular sets, unary sets, decidability

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.24

## 1 Introduction

We investigate separability problems for sets of vectors from  $\mathbb{N}^d$ . We say that a set  $U$  is *separated from* a set  $V$  by a set  $S$  if  $U \subseteq S$  and  $V \cap S = \emptyset$ . For two families of sets  $\mathcal{F}$  and  $\mathcal{G}$ , the  $\mathcal{F}$ -separability problem for  $\mathcal{G}$  asks for two given sets  $U, V \in \mathcal{G}$  whether  $U$  is separated from  $V$  by some set from  $\mathcal{F}$ . Concretely, we consider  $\mathcal{F}$  to be modular sets or unary sets<sup>1</sup>, and  $\mathcal{G}$  to be reachability sets of Vector Addition Systems or generalizations thereof.

**Motivation.** The separability problem is a classical problem in theoretical computer science. It was investigated most extensively in the area of formal languages, for  $\mathcal{G}$  being the family of all regular word languages. Since regular languages are effectively closed under complement, the  $\mathcal{F}$ -separability problem is a generalization of the  $\mathcal{F}$ -characterization problem, which asks whether a given language belongs to  $\mathcal{F}$ . Indeed,  $L \in \mathcal{F}$  if and only if  $L$  is separated from its complement by some language from  $\mathcal{F}$ . Separability problems for regular languages attracted recently a lot of attention, which resulted in establishing the decidability of  $\mathcal{F}$ -separability for

---

\* The first three authors have been partially supported by the NCN grant 2013/09/B/ST6/01575.

<sup>1</sup> Since  $S$  separates  $U$  from  $V$  iff its complement separates  $V$  from  $U$ , and since  $\mathcal{F}$  is closed under complement, we could equally well have defined a symmetric version of the separability problem by saying that  $S$  separates  $\{U, V\}$  iff  $U \subseteq S$  and  $V \cap S = \emptyset$ .



the family  $\mathcal{F}$  of separators being the piecewise testable languages [3, 21] (recently generalized to finite ranked trees [6]), the locally and locally threshold testable languages [20], the languages definable in first order logic [23], and the languages of certain higher levels of the first order hierarchy [22], among others.

Separability of nonregular languages attracted little attention till now. The reasons for this are twofold. First, for regular languages one can use standard algebraic tools, like syntactic monoids, and indeed most of the results have been obtained with the help of such techniques. Second, some strong intractability results have been known already since the 70's, when Szymanski and Williams proved that regular separability of context-free languages is undecidable [24]. Later Hunt [11] generalized this result: he showed that  $\mathcal{F}$ -separability of context-free languages is undecidable for every class  $\mathcal{F}$  which is closed under finite boolean combinations and contains all languages of the form  $w\Sigma^*$  for  $w \in \Sigma^*$ . This is a very weak condition, so it seemed that nothing nontrivial can be done outside regular languages with respect to separability problems. Furthermore, Kopczyński has recently shown that regular separability is undecidable even for languages of visibly pushdown automata [13], thus strengthening the result by Szymanski and Williams. On the positive side, piecewise testable separability has been shown decidable for context-free languages, languages of Vector Addition Systems (VAS languages), and some other classes of languages [4]. This inspired us to start a quest for decidable cases beyond regular languages.

To the best of our knowledge, beside [4], separability problems for VAS languages have not been investigated before.

**Our contribution.** In this paper, we get a step closer towards solving regular separability of VAS languages. Instead of VAS languages themselves (i.e., subsets of  $\Sigma^*$ ), in this paper we investigate their commutative closures, or, alternatively, subsets of  $\mathbb{N}^d$  represented as reachability sets of VASes, VASes with states, or Petri nets. A VAS reachability set is just the set of configurations of a VAS which can be reached from a specified initial configuration. Towards a unified treatment, instead of considering separately VASes, VASes with states, and Petri nets, we consider *sections* of VAS reachability sets (abbreviated as VAS sections below), which turn out to be expressive enough to represent sections of VASes with states and Petri nets, and thus being a convenient subsuming formalism. A *section* of a set of vectors  $X \subseteq \mathbb{N}^d$  is the set obtained by first fixing a value for certain coordinates, and then projecting the result to the remaining coordinates. For example, if  $X$  is the set of pairs  $\{(x, y) \in \mathbb{N}^2 \mid x \text{ divides } y\}$ , then the section of  $X$  obtained by fixing the first coordinate to 3 is the set  $\{0, 3, 6, \dots\}$ . It can be easily shown that VAS sections are strictly more general than VAS reachability sets themselves, and they are equiexpressive with sections of VASes with states and Petri nets.

We study the separability problem of VAS sections by simpler classes, namely, modular and unary sets. A set  $X \subseteq \mathbb{N}^d$  is *modular* if there exists a modulus  $n \in \mathbb{N}$  s.t.  $X$  is closed under the congruence modulo  $n$  on every coordinate, and it is *unary* if there exists a threshold  $n \in \mathbb{N}$  s.t. it is closed under the congruence modulo  $n$  above the threshold  $n$  on every coordinate. Clearly, VAS sections are more general than both unary and modular sets, and unary sets are more general than modular sets. Moreover, unary sets are tightly connected with commutative regular languages, in the sense that the Parikh image<sup>2</sup> of a commutative regular language is a unary set, and vice versa, the inverse Parikh image of a

---

<sup>2</sup> The Parikh image of a language of words  $L \subseteq \{a_1, \dots, a_k\}$  is the subset of  $\mathbb{N}^k$  obtained by counting occurrences of letters in  $L$ .

unary set is a commutative regular language. As our main result, we show that the modular and unary separability problems are decidable for VAS sections (and thus for sections of VASes with states and Petri nets). Both proofs use similar techniques, and invoke two semi-decision procedures: the first one (positive) enumerates witnesses of separability, and the second one (negative) enumerates witnesses of nonseparability. A separability witness is just a modular (or unary) set, and verifying that it is indeed a separator easily reduces to the VAS reachability problem. Thus, the hard part of the proof is to invent a finite and decidable witness of nonseparability, i.e., a finite object whose existence proves that none of infinitely many modular (resp. unary) sets is a separator. Our main technical observation is that two nonseparable VAS reachability sets always admit two *linear* subsets thereof that are already nonseparable.

From our result, thanks to the tight connection between unary sets and commutative regular languages mentioned above, we can immediately deduce decidability of regular separability for *commutative closures of VAS languages*, and *commutative regular separability* for VAS languages. This constitutes a first step towards determining the status of regular separability for languages of VASes. Full proofs can be found in the technical report [2].

**Related research.** Choffrut and Grigorieff have shown decidability of separability of rational relations by recognizable relations in  $\Sigma^* \times \mathbb{N}^d$  [1]. Rational subsets of  $\mathbb{N}^d$  are precisely the semilinear sets, and recognizable (by morphism into a monoid) subsets of  $\mathbb{N}^d$  are precisely the unary sets. Thus, by ignoring the  $\Sigma^*$  component, one obtains a very special case of our result, namely decidability of the unary separability problem for semilinear sets. Moreover, since modular sets are subsets of  $\mathbb{N}^d$  which are recognizable by a morphism into a monoid which happens to be a group, we also obtain a new result, namely, decidability of separability of rational subsets of  $\mathbb{N}^d$  by subsets of  $\mathbb{N}^d$  recognized by a group.

From a quite different angle, our research seems to be closely related to the VAS reachability problem. Leroux [16] has shown a highly nontrivial result: the reachability sets of two VASes are disjoint if, and only if, they can be separated by a semilinear set. In other words, semilinear separability for VAS reachability sets is equivalent to the VAS (non-)reachability problem. This connection suggests that modular and unary separability are interesting problems in themselves, enriching our understanding of VASes. Finally, we show that VAS reachability reduces to unary separability, thus the problem does not become easier by considering the simpler class of unary sets as opposed to semilinear sets. For modular separability we have a weaker complexity lower bound, i.e. EXPSPACE-hardness, by a reduction from control state reachability for VASSes.

## 2 Preliminaries

**Vectors.** By  $\mathbb{N}$  and  $\mathbb{Z}$  we denote the set of natural and integer numbers, respectively. For a vector  $u = (u_1, \dots, u_d) \in \mathbb{Z}^d$  and for a coordinate  $i \in \{1, \dots, d\}$ , we denote by  $u[i]$  its  $i$ -th component  $u_i$ . The zero vector is denoted by  $0$ . The order  $\leq$  and the sum operation  $+$  naturally extend to vectors pointwise. Moreover, if  $n \in \mathbb{Z}$ , then  $nu$  is the vector  $(nu_1, \dots, nu_d)$ . These operations extend to sets element-wise in the natural way: For two sets of vectors  $U, V \subseteq \mathbb{Z}^d$  we denote by  $U + V$  its Minkowski sum  $\{u + v \mid u \in U, v \in V\}$ . For a (possibly infinite) set of vectors  $S \subseteq \mathbb{Z}^d$ , let  $\text{LIN}(S)$  and  $\text{LIN}^{\geq 0}(S)$  be the set of *linear combinations* and *non-negative linear combinations* of vectors from  $S$ , respectively, i.e.,

$$\begin{aligned} \text{LIN}(S) &= \{a_1 v_1 + \dots + a_k v_k \mid v_1, \dots, v_k \in S, a_1, \dots, a_k \in \mathbb{Z}\}, \text{ and} \\ \text{LIN}^{\geq 0}(S) &= \{a_1 v_1 + \dots + a_k v_k \mid v_1, \dots, v_k \in S, a_1, \dots, a_k \in \mathbb{N}\}. \end{aligned}$$

When the set  $S = \{v_1, \dots, v_k\}$  is finite, we alternatively write  $\text{LIN}(v_1, \dots, v_k)$  instead of  $\text{LIN}(\{v_1, \dots, v_k\})$ , and similarly for  $\text{LIN}^{\geq 0}(v_1, \dots, v_k)$ .

**Modular, unary, linear, and semilinear sets.** Two vectors  $x, y \in \mathbb{Z}^d$  are *n-modular equivalent*, written  $x \equiv_n y$ , if, for all  $i \in \{1, \dots, d\}$ , we have  $x[i] \equiv y[i] \pmod n$ . Moreover, two non-negative vectors  $x, y \in \mathbb{N}^d$  are *n-unary equivalent*, written  $x \cong_n y$ , if  $x \equiv_n y$  and  $x[i] \geq n \iff y[i] \geq n$  for all  $i \in \{1, \dots, d\}$ . A  $d$ -dimensional set  $S \subseteq \mathbb{N}^d$  is *modular* if there exists a number  $n \in \mathbb{N}$ , s.t.  $S$  is a union of  $n$ -modular equivalence classes. *Unary* sets  $S \subseteq \mathbb{N}^d$  are defined similarly w.r.t.  $n$ -unary equivalence classes.

A set  $S \subseteq \mathbb{N}^d$  is *linear* if it is of the form  $S = \{b\} + \text{LIN}^{\geq 0}(p_1, \dots, p_k)$  for some *base*  $b \in \mathbb{N}^d$  and some *periods*  $p_1, \dots, p_k \in \mathbb{N}^d$ . A set is *semilinear* if it is a finite union of linear sets. Note that a modular set is also unary (since  $\cong_n$  is finer than  $\equiv_n$ ), and that unary set is in turn a semilinear set, which can be presented as a finite union of linear sets in which all the periods are parallel to the coordinate axes, i.e., they have exactly one non-zero entry.

**Separability.** For  $S, U, V \subseteq \mathbb{N}^d$ , we say that  $S$  *separates*  $U$  from  $V$  if  $U \subseteq S$  and  $V \cap S = \emptyset$ . The set  $S$  is also called a *separator* of  $U, V$ . For a family  $\mathcal{F}$  of sets, we say that  $U$  is  $\mathcal{F}$  *separable* from  $V$  if  $U$  is separated from  $V$  by a set  $S \in \mathcal{F}$ . In this paper, the set of separators  $\mathcal{F}$  will be the modular sets and the unary ones. Since both classes are closed under complement, the notion of  $\mathcal{F}$  separability is symmetric:  $U$  is  $\mathcal{F}$  separable from  $V$  iff  $V$  is  $\mathcal{F}$  separable from  $U$ . Thus we use also a symmetric notation, in particular we say that  $U$  and  $V$  are  $\mathcal{F}$  separable instead of saying that  $U$  is  $\mathcal{F}$  separable from  $V$ . For two families of sets  $\mathcal{F}$  and  $\mathcal{G}$ , the  *$\mathcal{F}$  separability problem for  $\mathcal{G}$*  asks whether two given sets  $U, V \in \mathcal{G}$  are  $\mathcal{F}$  separable. In this paper we mainly consider two instances of  $\mathcal{F}$ , namely modular sets and unary sets, and thus we speak of *modular* and *unary separability* problems, respectively.

**Vector Addition Systems.** A  $d$ -dimensional *Vector Addition System* (VAS) is a pair  $V = (s, T)$ , where  $s \in \mathbb{N}^d$  is the *source* configuration and  $T \subseteq_{\text{FIN}} \mathbb{Z}^d$  is the set of finitely many *transitions*. A *partial run*  $\rho$  of a VAS  $V = (s, T)$  is a sequence  $(v_0, t_0, v_1), \dots, (v_{n-1}, t_{n-1}, v_n) \in \mathbb{N}^d \times T \times \mathbb{N}^d$  such that for all  $i \in \{0, \dots, n-1\}$  we have  $v_i + t_i = v_{i+1}$ . The *source* of this partial run is the configuration  $v_0$  and the *target* of this partial run is the configuration  $v_n$ , we write  $\text{SOURCE}(\rho) = v_0$ ,  $\text{TARGET}(\rho) = v_n$ . The *labeling* of  $\rho$  is the sequence  $t_0 \dots t_{n-1} \in T^*$ , we write  $\text{LABEL}(\rho) = t_0 \dots t_{n-1}$ . For a sequence  $\alpha \in T^*$  and a partial run  $\rho$  such that  $\text{LABEL}(\rho) = \alpha$ ,  $\text{SOURCE}(\rho) = u$  and  $\text{TARGET}(\rho) = v$  we write  $u \xrightarrow{\alpha} v$  to denote this unique partial run. A partial run  $\rho$  of  $(s, T)$  with  $\text{SOURCE}(\rho) = s$  is called a *run*. The set of all runs of a VAS  $V$  is denoted as  $\text{RUNS}(V)$ . The *reachability set*  $\text{REACH}(V)$  of a VAS  $V$  is the set of targets of all its runs; the sets  $\text{REACH}(V)$  we call *VAS reachability sets* in the sequel. The family of all VAS reachability sets we denote as  $\text{REACH}(\text{VAS})$ .

► **Example 1.** Consider a VAS  $V = (s, T)$ , for a source configuration  $s = (1, 0, 0)$  and a set of transitions  $T = \{(-1, 2, 1), (2, -1, 1)\}$ . One easily proves that

$$\text{REACH}(V) = \{(a, b, c) \in \mathbb{N}^3 \mid a + b = c + 1 \wedge a - b \equiv 1 \pmod 3\}.$$

**Vector Addition Systems with states.** A  $d$ -dimensional *VAS with states* (VASS) is a triple  $V = (s, T, Q)$ , where  $Q$  is a finite set of *states*,  $s \in Q \times \mathbb{N}^d$  is the *source* configuration and  $T \subseteq_{\text{FIN}} Q \times \mathbb{Z}^d \times Q$  is a finite set of *transitions*. Similarly as in case of VASes, a *run*  $\rho$  of a VASS  $V = (s, T, Q)$  is a sequence  $(q_0, v_0, s_0, q_1, v_1), \dots, (q_{n-1}, v_{n-1}, s_{n-1}, q_n, v_n) \in Q \times \mathbb{N}^d \times \mathbb{Z}^d \times Q \times \mathbb{N}^d$  such that  $(q_0, v_0) = s$  and for all  $i \in \{0, \dots, n-1\}$  we have



$(q_i, s_i, q_{i+1}) \in T$  and  $v_i + s_i = v_{i+1}$ . We write  $\text{TARGET}(\rho) = (q_n, v_n)$ . The *reachability set* of a VASS  $V$  in state  $q$  is  $\text{REACH}_q(V) = \{v \in \mathbb{N}^d \mid (q, v) = \text{TARGET}(\rho) \text{ for some run } \rho\}$ . The family of all such reachability sets of all VASSes we denote as  $\text{REACH}(\text{VASS})$ .

► **Example 2** (cf. [9]). Let  $V$  be a 3-dimensional VASS with two states,  $p$  and  $p'$ , the source configuration  $(p, (1, 0, 0))$ , and four transitions:

$$(p, (-1, 1, 0), p), \quad (p, (0, 0, 0), p'), \quad (p', (2, -1, 0), p'), \quad (p', (0, 0, 1), p).$$

Then  $\text{REACH}_p(V) = \{(a, b, c) \in \mathbb{N}^3 \mid 1 \leq a + b \leq 2^c\}$ .

### 3 Sections

VAS reachability sets are central for this paper. However, in order to make this family of sets more robust, we prefer to consider the slightly larger family of *sections* of VAS reachability sets. The intuition about a section is that we fix values on a subset of coordinates in vectors, and collect all the values that can occur on the other coordinates. For a vector  $u \in \mathbb{N}^d$  and a subset  $I \subseteq \{1, \dots, d\}$  of coordinates, by  $\pi_I(u) \in \mathbb{N}^{|I|}$  we denote the *I-projection* of  $u$ , i.e., the vector obtained from  $u$  by removing coordinates not belonging to  $I$ . The projection extends element-wise to sets of vectors  $S \subseteq \mathbb{N}^d$ , denoted  $\pi_I(S)$ . For a set of vectors  $S \subseteq \mathbb{N}^d$ , a subset  $I \subseteq \{1, \dots, d\}$ , and a vector  $u \in \mathbb{N}^{d-|I|}$ , the *section* of  $S$  w.r.t.  $I$  and  $u$  is the set

$$\text{SEC}_{I,u}(S) := \pi_I(\{v \in S \mid \pi_{\{1,\dots,d\} \setminus I}(v) = u\}) \subseteq \mathbb{N}^{|I|}.$$

We denote by  $\text{SECREACH}(\text{VAS})$  the family of all sections of VAS reachability sets, which we abbreviate as *VAS sections below*. Similarly, the family of all sections of VASS-reachability sets we denote by  $\text{SECREACH}(\text{VASS})$ .

► **Example 3.** Consider the VAS  $V$  from Example 1. For  $I = \{1, 2\}$  and  $u = 7 \in \mathbb{N}^1$  we have  $\text{SEC}_{I,u}(\text{REACH}(V)) = \{(0, 8), (3, 5), (6, 2)\}$ .

Note that in a special case of  $I = \{1, \dots, d\}$ , when  $u$  is necessarily the empty vector,  $\text{SEC}_{I,u}(S) = S$ . Thus  $\text{REACH}(\text{VAS})$  is a subfamily of  $\text{SECREACH}(\text{VAS})$ , and likewise for VASSes. We argue that VAS sections are a more robust class than VAS reachability sets. Indeed, as shown below VAS sections are closed under positive boolean combinations, which is not the case for VAS reachability sets.

Reachability sets of VASes are a strict subfamily of reachability sets of VASes with states, which in turn are a strict subfamily of sections of reachability sets of VASes. However, when sections of reachability set are compared, there is no difference between VASes and VASes with states, which motivates considering sections in this paper. These observations are summarized in the following two propositions:

► **Proposition 4.**  $\text{REACH}(\text{VAS}) \subsetneq \text{REACH}(\text{VASS}) \subsetneq \text{SECREACH}(\text{VAS})$ .

► **Proposition 5.**  $\text{SECREACH}(\text{VAS}) = \text{SECREACH}(\text{VASS})$ .

► **Remark.** In a similar vein one shows that reachability sets of Petri nets include  $\text{REACH}(\text{VAS})$  and are included in  $\text{REACH}(\text{VASS})$ . Therefore, as long as sections are considered, there is no difference between VASes, Petri nets, and VASSes. In consequence, our results apply not only to VASes, but to all the three models.

We conclude this section by stating closure property of VAS sections. By *positive boolean combination* we mean sets obtained by taking only intersections and unions, but not complements.

► **Proposition 6.** *The family of VAS sections is closed under positive boolean combinations.*

## 4 Results

As our main technical contribution, we prove decidability of the modular and unary separability problems for the class of sections of VAS reachability sets.

► **Theorem 7.** *The modular separability problem for VAS sections is decidable.*

► **Theorem 8.** *The unary separability problem for VAS sections is decidable.*

The proofs are postponed to Sections 5–7. Furthermore, as a corollary of Theorem 8 we derive decidability of two commutative variants of the regular separability of VAS languages (formulated in Theorems 9 and 10 below).

To consider languages instead of reachability sets, we need to assume that transitions of a VAS are *labeled* by elements of an alphabet  $\Sigma$ , and thus every run is labeled by a word over  $\Sigma$  obtained by concatenating labels of consecutive transitions of a run. We allow for silent transitions labeled by  $\varepsilon$ , i.e., transitions that do not contribute to the labeling of a run. The language  $L(V)$  of a VAS  $V$  contains labels of those runs of  $V$  that end in an *accepting* configuration. Our results work for several variants of acceptance; for instance, for a fixed configuration  $v_0$ , we may consider a configuration  $v$  accepting if:

- $v \geq v_0$ , this choice yields the so called *coverability languages*; or
- $v = v_0$ , this choice yields *reachability languages*.

The Parikh image of a word  $w \in \Sigma^*$ , for a fixed total ordering  $a_1 < \dots < a_d$  of  $\Sigma$ , is a vector in  $\mathbb{N}^d$  whose  $i$ th coordinate stores the number of occurrences of  $a_i$  in  $w$ . We lift the operation element-wise to languages, thus the Parikh image of a language  $L$ , denoted  $\Pi(L)$ , is a subset of  $\mathbb{N}^d$ . Two words  $w, v$  over  $\Sigma$  are *commutatively equivalent* if their Parikh images are equal. The *commutative closure* of a language  $L \subseteq \Sigma^*$ , denoted  $\text{CC}(L)$ , is the language containing all words  $w \in \Sigma^*$  commutatively equivalent to some word  $v \in L$ . A language  $L$  is *commutative* if it is invariant under commutative equivalence, i.e.,  $L = \text{CC}(L)$ . Note that a commutative language is uniquely determined by its Parikh image. The Parikh image of any commutative regular language is unary: A finite automaton recognizing a commutative language can only count modulo  $n$  above threshold  $n$  for each letter in the alphabet independently. Moreover, all unary set can be obtained as the Parikh image of a commutative regular language. Similarly, the inverse Parikh image of a unary set is a commutative regular language, and all commutative regular languages can be obtained in this way. In this sense, commutative regular languages and unary sets are in correspondence with each other.

As a corollary of Theorem 8 we deduce decidability of the following two commutative variants of the regular separability of VAS languages:

- *commutative regular separability of VAS languages*: given two VASes  $V, V'$ , decide whether there is a commutative regular language  $R$  that includes  $L(V)$  and is disjoint from  $L(V')$ ;
- *regular separability for commutative closures of VAS languages*: given two VASes  $V, V'$ , decide whether there is a regular language  $R$  that includes  $\text{CC}(L(V))$  and is disjoint from  $\text{CC}(L(V'))$ .

► **Theorem 9.** *Commutative regular separability is decidable for VAS languages.*

Indeed, given two VASes  $V, W$  one easily constructs another two VASes  $V', W'$  s.t.  $\Pi(L(V))$  is a section of  $\text{REACH}(V')$ , and similarly for  $W'$ . By the tight correspondence between commutative regular languages and unary sets, we observe that  $L(V)$  and  $L(W)$  are separated by a commutative regular language if, and only if, their Parikh images  $\Pi(L(V))$  and  $\Pi(L(W))$  are separated by a unary set, which is decidable by Theorem 8.

► **Theorem 10.** *Regular separability is decidable for commutative closures of VAS languages.*

Similarly as above, we reduce to unary separability of VAS reachability sets (which is decidable once again by Theorem 8), which is immediate once one proves the following crucial observation.

► **Lemma 11.** *Two commutative languages  $L, K \subseteq \Sigma^*$  are regular separable if, and only if, their Parikh images are unary separable.*

**Proof.** For the “if” direction, let  $\Pi(K)$  and  $\Pi(L)$  be separable by some unary set  $U \subseteq \mathbb{N}^d$ . Let  $S = \{w \in \Sigma^* \mid \Pi(w) \in U\}$ . It is easy to see that  $S$  is (commutative) regular since  $U$  is unary, and that  $S$  separates  $K$  and  $L$ . For the “only if” direction, let  $K$  and  $L$  be separable by a regular language  $S$ , say  $K \subseteq S$  and  $S \cap L = \emptyset$ . Let  $M$  be the syntactic monoid of  $S$  and  $\omega$  be its idempotent power, i.e., a number such that  $m^\omega = m^{2\omega}$  for every  $m \in M$ . In particular, for every word  $u \in \Sigma^*$  we have (P)  $uv^\omega w \in S \iff uv^{2\omega} w \in S$ ; in other words, one can substitute  $v^\omega$  by  $v^{2\omega}$  and vice versa in every context. Let  $\Sigma = \{a_1, \dots, a_d\}$ . For  $u = (u_1, \dots, u_d) \in \mathbb{N}^d$  define the word  $w_u = a_1^{u_1} \dots a_d^{u_d}$ . For every  $u, v \in \mathbb{N}^d$  such that  $u \cong_\omega v$ , by repetitive application of (P) we get  $w_u \in S$  iff  $w_v \in S$ . As  $K$  is commutative and  $K \subseteq S$ , we have  $w_u \in S$  for all  $u \in \Pi(K)$ ; similarly, we have  $w_v \notin S$  for all  $v \in \Pi(L)$ . Thus, for all  $u \in \Pi(K)$ ,  $v \in \Pi(L)$  we have  $u \not\cong_\omega v$ . Let  $U = \{x \in \mathbb{N}^d \mid \exists y \in \Pi(K) x \cong_\omega y\}$ . The set  $U$  separates  $\Pi(K)$  and  $\Pi(L)$  and, being a union of  $\cong_\omega$  equivalence classes, it is unary. ◀

## 5 Modular separability of linear sets

The rest of the paper is devoted to the proofs of Theorems 7 and 8. In this section we prove two preliminary results that will be later used in Section 6, where the proof of Theorems 7 is completed. First, we prove a combinatorial result on linear combinations (cf. Lemma 12 below). Second, we prove that modular separability of linear sets is decidable (cf. Corollary 15). While this second result follows from [1] and is thus not a new result, we provide here another simple proof to make the paper self-contained.

**Linear combinations modulo  $n$ .** We start with some preliminary results from linear algebra. For  $n \in \mathbb{N}$ , let  $\text{LIN}_n^{\geq 0}(v_1, \dots, v_k)$  be the closure of  $\text{LIN}^{\geq 0}(v_1, \dots, v_k)$  modulo  $n$ , i.e.,

$$\text{LIN}_n^{\geq 0}(v_1, \dots, v_k) = \{v \in \mathbb{N}^d \mid \exists u \in \text{LIN}^{\geq 0}(v_1, \dots, v_k) v \equiv_n u\}.$$

Similarly one defines  $\text{LIN}_n(v_1, \dots, v_k)$  be the closure of  $\text{LIN}(v_1, \dots, v_k)$  modulo  $n$ . Observe however that  $\text{LIN}_n(v_1, \dots, v_k) = \text{LIN}_n^{\geq 0}(v_1, \dots, v_k)$ . Indeed, if  $v \equiv_n l_1 v_1 + \dots + l_k v_k$  for  $l_1, \dots, l_k \in \mathbb{Z}$  then  $v \equiv_n (l_1 + Nn)v_1 + \dots + (l_k + Nn)v_k$  for any  $N \in \mathbb{N}$ . The following observation connects linear combinations to the modular closure of non-negative linear combinations.

► **Lemma 12.**  $\text{LIN}(v_1, \dots, v_k) = \bigcap_{n>0} \text{LIN}_n^{\geq 0}(v_1, \dots, v_k)$ .

**Modular separability.** In the rest of the paper, we heavily rely on the following straightforward characterization of modular separability.

► **Proposition 13.** *Two sets  $U, V \subseteq \mathbb{N}^d$  are modular separable if, and only if, there exists  $n \in \mathbb{N}$  such that for all  $u \in U$ ,  $v \in V$  we have  $u \not\equiv_n v$ .*

In the special case of linear sets, the characterization above boils down to the following property:

► **Lemma 14.** *Two linear sets  $\{b\} + \text{LIN}^{\geq 0}(P)$  and  $\{c\} + \text{LIN}^{\geq 0}(Q)$  are not modular separable if, and only if,  $b - c \in \text{LIN}(P \cup Q)$ .*

Since the condition in the lemma above is effectively testable being an instance of solvability of systems of linear Diophantine equations, we get the following corollary:

► **Corollary 15.** *Modular separability of linear sets is decidable.*

► **Remark.** Since linear Diophantine equations are solvable in polynomial time, we obtain the same complexity for modular separability of linear sets. This observation however will not be useful in the sequel.

► **Remark.** The unary separability problem is decidable for linear sets, as shown in [1], but we will not need this fact in the sequel. Moreover, it follows from our stronger decidability result stated in Theorem 8, since linear sets are included in VAS sections.

## 6 Modular separability of VAS sections

In this section we prove Theorem 7, and thus provide an algorithm to decide modular separability for VAS reachability sets. Given two VAS sections  $U$  and  $V$ , the algorithm runs in parallel two semi-decision procedures: one (positive) which looks for a witness of separability, and another one (negative) which looks for a witness of nonseparability. Directly from the characterization of Proposition 13, the positive semi-decision procedure simply enumerates all moduli  $n \in \mathbb{N}$  and checks whether  $u \not\equiv_n v$  for all  $u \in U$  and  $v \in V$ . The latter condition can be decided by reduction to the VAS (non)reachability problem [19, 17].

► **Lemma 16.** *For two VAS sections  $U$  and  $V$  and a modulus  $n \in \mathbb{N}$ , it is decidable whether there exist  $u \in U$  and  $v \in V$  s.t.  $u \equiv_n v$ .*

It remains to design the negative semi-decision procedure, which is the nontrivial part. In Lemma 22, we show that if two VAS reachability sets are not modular separable, then in fact they already contain two *linear* subsets which are not modular separable. In order to construct such linear witnesses of nonseparability, we use the theory of well quasi orders and some elementary results in algebra, which we present next.

**The order on runs.** In this section, we define a certain well quasi order on runs  $\preceq$  which will prove useful in the following; a weaker version of this order was defined in [12].

A quasi order  $(X, \preceq)$  is a *well quasi order* (wqo) if for every infinite sequence  $x_0, x_1, \dots \in X$  there exist indices  $i, j \in \mathbb{N}, i < j$ , such that  $x_i \preceq x_j$ . It is folklore that if  $(X, \preceq)$  is a wqo, then in every infinite sequence  $x_0, x_1, \dots \in X$  there even exists an infinite monotonically non-decreasing subsequence  $x_{i_1} \preceq x_{i_2} \preceq \dots$ . We will use Dickson's [5] and Higman's [8] Lemmas to define new wqo's on pairs and sequences. For two qos  $(X, \leq_X)$  and  $(Y, \leq_Y)$ , let the product  $(X \times Y, \leq_{X \times Y})$  be ordered componentwise by  $(x, y) \leq_{X \times Y} (x', y')$  if  $x \leq_X x'$  and  $y \leq_Y y'$ .

► **Lemma 17** (Dickson [5]). *If  $(X, \leq_X)$  and  $(Y, \leq_Y)$  are wqos, then  $(X \times Y, \leq_{X \times Y})$  is a wqo.*

As a corollary, if two qos  $(X, \leq_1)$  and  $(X, \leq_2)$  on the same domain are wqos, then also their intersection is a wqo. For a qo  $(X, \leq)$ , let  $(X^*, \leq_*)$  be quasi ordered by the subsequence order  $\leq_*$ , defined as  $x_1 x_2 \dots x_k \leq_* y_1 y_2 \dots y_m$  if there exist  $1 \leq i_1 < \dots < i_k \leq m$  such that  $x_j \leq y_{i_j}$  for all  $j \in \{1, \dots, k\}$ .

► **Lemma 18** (Higman [8]). *If  $(X, \leq)$  is a wqo then  $(X^*, \leq_*)$  is a wqo.*

By considering the finite set of transitions  $T$  well quasi ordered by equality, we define the order  $\leq^1$  on triples  $\mathbb{N}^d \times T \times \mathbb{N}^d$  componentwise as  $(u, s, u') \leq^1 (v, t, v')$  if  $u \leq v$ ,  $s = t$ , and  $u' \leq v'$ , which is a wqo by Dickson's Lemma. We further extend  $\leq^1$  to an order  $\preceq$  on runs by defining, for two runs  $\rho$  and  $\sigma$  in  $(\mathbb{N}^d \times T \times \mathbb{N}^d)^*$ ,  $\rho \preceq \sigma$  if  $\rho \leq_*^1 \sigma$  and  $\text{TARGET}(\rho) \leq \text{TARGET}(\sigma)$ ; Here,  $\leq_*^1$  is the extension of  $\leq^1$  to sequences, and thus a wqo by Higman's Lemma, which implies that  $\preceq$  is itself a wqo. Our order  $\preceq$  is very similar to the weaker order defined in [12], which is the same as  $\preceq$ , except that it does not include target configurations.

► **Proposition 19.**  $\preceq$  is a well quasi order.

The following lemma is a quantitative version of monotonicity of VASes, and it says not only that larger runs can do more than smaller runs, but also that all nonnegative linear combinations of increments of larger runs can in fact be realized.

► **Lemma 20.** Let  $\rho_0, \rho_1, \dots, \rho_k$  be runs of a VAS s.t., for all  $i \in \{1, \dots, k\}$ ,  $\rho_0 \preceq \rho_i$ , and let  $\delta_i := \text{TARGET}(\rho_i) - \text{TARGET}(\rho_0) \geq 0$ . For any  $\delta \in \text{LIN}^{\geq 0}(\delta_1, \dots, \delta_k)$ , there exists a run  $\rho$  s.t.  $\rho_0 \preceq \rho$  and  $\delta = \text{TARGET}(\rho) - \text{TARGET}(\rho_0)$ .

We conclude this part by showing that any (possibly infinite) subset of  $\mathbb{Z}^d$  can be overapproximated by taking linear combinations of a *finite* subset thereof. This will be important below in order to construct linear sets as witnesses of nonseparability.

► **Lemma 21.** For every (possibly infinite) set of vectors  $S \subseteq \mathbb{Z}^d$ , there exist finitely many vectors  $v_1, \dots, v_k \in S$  s.t.  $S \subseteq \text{LIN}(v_1, \dots, v_k)$ .

**Proof.** Treat  $\mathbb{Z}^d$  as a freely finitely generated abelian group, and consider the subgroup  $\text{LIN}(S)$  of  $\mathbb{Z}^d$  generated by  $S$ , i.e., the subgroup containing all linear combinations of finitely many elements of  $S$ . We use the following result in algebra: every subgroup of a finitely generated abelian group is finitely generated (see for instance Corollary 1.7, p. 74, in [10]). By this result applied to  $\text{LIN}(S)$  we get a finite set of generators  $F \subseteq \text{LIN}(S)$  s.t.  $\text{LIN}(F) = \text{LIN}(S)$ . Every element of  $F$  is a linear combination of finitely many elements of  $S$ . Thus let  $v_1, \dots, v_k$  be all the elements of  $S$  appearing as a linear combination of some element from  $F$ . Then clearly  $F \subseteq \text{LIN}(v_1, \dots, v_k)$ , and thus  $S \subseteq \text{LIN}(S) = \text{LIN}(F) \subseteq \text{LIN}(\text{LIN}(v_1, \dots, v_k)) = \text{LIN}(v_1, \dots, v_k)$ , as required. ◀

► **Remark.** In fact one can show that the generating set  $F$  has at most  $d$  elements. However, no upper bound on  $k$  follows, and even for  $d = 1$  the number of vectors  $k$  can be arbitrarily large. Indeed, let  $p_1, \dots, p_k$  be different prime numbers, let  $u_i = (p_1 \cdot \dots \cdot p_k) / p_i$  and  $S = \{u_1, \dots, u_k\}$ . Then for every  $i \in \{1, \dots, k\}$ , the number  $u_i$  is not a linear combination of numbers  $u_j$ ,  $j \neq i$ , as  $u_i$  is not divisible by  $p_i$ , while all the others are. Therefore we need all the elements of  $S$  in the set  $\{v_1, \dots, v_k\}$ .

**Modular nonseparability witness.** We now concentrate on the negative semi-decision procedure. Let  $U, V \subseteq \mathbb{N}^d$  be two VAS sections:

$$U = \text{SEC}_{I, \bar{u}}(R_U) \subseteq \mathbb{N}^d \quad \text{and} \quad V = \text{SEC}_{J, \bar{v}}(R_V) \subseteq \mathbb{N}^d,$$

where  $R_U \subseteq \mathbb{N}^{d_U}$  and  $R_V \subseteq \mathbb{N}^{d_V}$  are the reachability sets of the two VASes  $W_U$  and  $W_V$ , and  $I \subseteq \{1, \dots, d_U\}$  and  $J \subseteq \{1, \dots, d_V\}$  with  $|I| = |J| = d$  are projecting coordinates, and  $\bar{u} \in \mathbb{N}^{d_U-d}$ ,  $\bar{v} \in \mathbb{N}^{d_V-d}$  are two sectioning vectors.

Observe that by padding coordinates we can assume w.l.o.g. that the two input VASes have the same dimension  $d' = d_U = d_V$ . Furthermore, we can also assume w.l.o.g. that

## 24:10 Separability of Reachability Sets of Vector Addition Systems

$\bar{u} = \bar{v} = 0$ . Indeed, one can add an additional coordinate, such that for performing any transition it is necessary that this coordinate is nonzero and a special, final transition, which causes the additional coordinate to be equal zero and subtracts  $\bar{u}$  (or  $\bar{v}$ ) from the other coordinates. The result of adding this gadget is that now we can assume  $\bar{u} = \bar{v} = 0$ , but the section itself does not change.

Finally, by reordering coordinates we can guarantee that the coordinates that are projected away appear on the same positions in both VASes, i.e.,  $I = J$ . With these assumptions, we observe that modular separability of sets  $U, V \subseteq \mathbb{N}^d$  is equivalent to modular separability of sets  $U', V' \subseteq \mathbb{N}^{d'}$ , defined as  $U, V$  but *without* projecting onto the subset  $I$  of coordinates:

$$U' = \{v \in R_U \mid \pi_{\{1, \dots, d'\} \setminus I}(v) = 0\} \quad V' = \{v \in R_V \mid \pi_{\{1, \dots, d'\} \setminus I}(v) = 0\}.$$

We call the set  $U'$  (resp.  $V'$ ) the *expansion* of  $U$  (resp.  $V$ ).

We say that a linear set  $L = \{b\} + \text{LIN}^{\geq 0}(p_1, \dots, p_k) \subseteq \mathbb{N}^{d'}$  is a *U-witness* if  $W_U$  admits runs  $\rho, \rho_1, \dots, \rho_k$  s.t.  $\rho \preceq \rho_1, \dots, \rho \preceq \rho_k$ , and

$$\begin{aligned} b &= \text{TARGET}(\rho) \in U' \\ b + p_i &= \text{TARGET}(\rho_i) \in U' \quad \text{for } i \in \{1, \dots, k\} \end{aligned} \tag{1}$$

Analogously one defines *V-witnesses*, but with respect to  $W_V$ .

► **Lemma 22.** *For two VAS sections  $U, V \subseteq \mathbb{N}^d$ , the following conditions are equivalent:*

1.  $U, V$  are not modular separable;
2. the expansions  $U', V'$  of  $U, V$  are not modular separable;
3. there exist linear subsets  $L \subseteq U', M \subseteq V'$  that are not modular separable;
4. there exist a *U-witness*  $L$  and a *V-witness*  $M$  that are not modular separable.

**Proof.** Equivalence of points 1 and 2 follows by the definition of expansion. Point 4 implies 3, as a *U-witness* is necessarily a subset of the expansion  $U'$  by Lemma 20. Point 3 implies 2, since if two sets are separable, also subsets thereof are separable (moreover, the separator remains the same). It remains to show that 2 implies 4.

Let  $U', V' \subseteq \mathbb{N}^{d'}$  be the expansions of two VAS sections  $U, V \subseteq \mathbb{N}^d$ , as above, and assume that they are not modular separable. We construct two linear sets  $L, M \subseteq \mathbb{N}^{d'}$  constituting a *U-witness* and a *V-witness*, respectively. By Proposition 13, there exists an infinite sequence of pairs of reachable configurations  $(u_0, v_0), (u_1, v_1), \dots \in U' \times V'$  s.t.  $u_n \equiv_n v_n$  for all  $n \in \mathbb{N}$ . By taking an appropriate infinite subsequence we can ensure that even  $u_n \equiv_n v_n$  for all  $n \in \mathbb{N}$ . Let us fix for every  $n \in \mathbb{N}$  runs  $\rho_n$  and  $\sigma_n$  such that  $u_n = \text{TARGET}(\rho_n)$  and  $v_n = \text{TARGET}(\sigma_n)$ . Since  $\preceq$  is a wqo by Proposition 19, we can extract a monotone non-decreasing subsequence, and thus we can ensure that even  $\rho_0 \preceq \rho_1 \preceq \dots$  and  $\sigma_0 \preceq \sigma_1 \preceq \dots$ . Here we use the fact that  $u_n \equiv_n v_n$  in the original sequence, and thus  $u_n \equiv_i v_n$  for every  $i \in \{1, \dots, n\}$ , consequently the new subsequence still has  $u_n \equiv_n v_n$  for all  $n \in \mathbb{N}$ . For all  $n \in \mathbb{N}$ , let  $\delta_n := u_n - u_0$  and  $\gamma_n := v_n - v_0$ , and consider the set of corresponding differences  $S_{\text{inf}} := \{\delta_n - \gamma_n \mid n \in \mathbb{N}\}$ . By Lemma 21, there exists a finite subset thereof  $S := \{\delta_{i_1} - \gamma_{i_1}, \dots, \delta_{i_k} - \gamma_{i_k}\}$  such that  $S_{\text{inf}} \subseteq \text{LIN}(S)$ , and thus there exist two finite subsets  $P := \{\delta_{i_1}, \dots, \delta_{i_k}\}$  and  $Q := \{\gamma_{i_1}, \dots, \gamma_{i_k}\}$  such that

$$S_{\text{inf}} \subseteq \text{LIN}(P - Q) \subseteq \text{LIN}(P) - \text{LIN}(Q) \subseteq \text{LIN}_n^{\geq 0}(P) - \text{LIN}_n^{\geq 0}(Q), \tag{2}$$

where the last inclusion follows from Lemma 12. Let  $L$  and  $M$  be defined as

$$L := \{u_0\} + \text{LIN}_n^{\geq 0}(P) \quad \text{and} \quad M := \{v_0\} + \text{LIN}_n^{\geq 0}(Q).$$

By construction,  $L$  is a  $U$ -witness and  $M$  a  $V$ -witness. It remains to show that  $L$  and  $M$  are not modular separable. For any  $n$ , by Eq. 2 we have  $\delta_n - \gamma_n \equiv_n \delta'_n - \gamma'_n$  for some  $\delta'_n \in \text{LIN}^{\geq 0}(P)$  and  $\gamma'_n \in \text{LIN}^{\geq 0}(Q)$ . Consider now the two new infinite sequences  $u'_1, u'_2, \dots \in L$  and  $v'_1, v'_2, \dots \in M$  defined as  $u'_n := u_0 + \delta'_n$  and  $v'_n := v_0 + \gamma'_n$ . Then,

$$\begin{aligned} u'_n - v'_n &= (u_0 + \delta'_n) - (v_0 + \gamma'_n) \\ &= (u_0 - v_0) + (\delta'_n - \gamma'_n) && \text{(by def. of } \delta'_n, \gamma'_n) \\ &\equiv_n (u_0 - v_0) + (\delta_n - \gamma_n) \\ &= (u_0 + \delta_n) - (v_0 + \gamma_n) \\ &= u_n - v_n \equiv_n 0 && \text{(by def. of } u_n, v_n), \end{aligned}$$

and thus  $u'_n \equiv_n v'_n$ . This, thanks to the characterization of Proposition 13, implies that  $L$  and  $M$  are not modular separable.  $\blacktriangleleft$

► **Remark.** Note that a modular nonseparability witness exists even in the case when the two reachability sets  $U, V$  have nonempty intersection. In this case, it is enough to consider two runs  $\rho_0$  and  $\sigma_0$  ending up in the same configuration  $\text{TARGET}(\rho_0) = \text{TARGET}(\sigma_0)$ , and considering the linear sets  $L := M := \{\text{TARGET}(\rho_0)\}$ .

Using the characterization of Lemma 22, the negative semi-decision procedure enumerates all pairs  $L, M$ , where  $L$  is a  $U$ -witness and  $M$  is a  $V$ -witness and checks whether  $L$  and  $M$  are modular separable, which is decidable due to Corollary 15. Enumerating  $U$ -witnesses (and  $V$ -witnesses) amounts of enumerating finite sets of runs  $\{\rho, \rho_1, \dots, \rho_k\}$  satisfying (1).

► **Remark.** It is also possible to design another negative semi-decision procedure using Lemma 22. This one enumerates all linear sets  $L$  and  $M$  (not necessarily only those in the special form of  $U$ - or  $V$ -witnesses) and checks whether they are modular separable *and* included in  $U$  and  $V$ , respectively. While this procedure is conceptually simpler than the one we presented, we now need the two extra inclusion checks  $L \subseteq U$  and  $M \subseteq V$ . Indeed,  $U$ - and  $V$ -witnesses were designed in such a way that the two inclusions above hold by construction and do not have to be checked. The problem whether a given linear set is included in a given VAS reachability set is decidable [15], however we chose to present the previous semi-decision procedure in order to be self contained.

## 7 Unary separability of VAS sections

The proof of Theorem 8 goes along the same lines as the proof of Theorem 7. It uses an immediate characterization of unary separability, which is the same as Proposition 13, with unary equivalence  $\cong_n$  in place of modular equivalence  $\equiv_n$ .

► **Proposition 23.** *Two sets  $U, V \subseteq \mathbb{N}^d$  are unary separable if, and only if, there exists  $n \in \mathbb{N}$  such that, for all  $u \in U$  and  $v \in V$ , we have  $u \not\cong_n v$ .*

As before, basing on the characterization of Proposition 23, the positive semi-decision procedure enumerates all  $n \in \mathbb{N}$  and checks whether the  $\cong_n$ -closures of the two reachability sets are disjoint, which is effective thanks to the following fact:

► **Lemma 24.** *For two VAS sections  $U$  and  $V$  and  $n \in \mathbb{N}$ , it is decidable whether there exist  $u \in U$  and  $v \in V$  such that  $u \cong_n v$ .*

This can be proved in a way similar to Lemma 16, with the adjustment that we allow on every coordinate a decrement by  $n$  only if the value is above  $2n$ . The negative semi-decision procedure enumerates nonseparability witnesses, and bases on the exact copy of Lemma 22, except that “modular” is replaced by “unary”:

► **Lemma 25.** *For two VAS sections  $U, V \subseteq \mathbb{N}^d$ , the following conditions are equivalent:*

1.  $U, V$  are not unary separable;
2. the expansions  $U', V'$  of  $U, V$  are not unary separable;
3. there exist linear subsets  $L \subseteq U', M \subseteq V'$  that are not unary separable;
4. there exist a  $U$ -witness  $L$  and a  $V$ -witness  $M$  that are not unary separable.

## 8 Final remarks

We have shown decidability of modular and unary separability for sections of VAS reachability sets, which include (sections of) reachability sets of VASes with states and Petri nets. As a corollary, we have derived decidability of regular separability of commutative closures of VAS languages, and of commutative regular separability of VAS languages. The decidability status of regular separability for VAS languages remains an intriguing open problem.

**Complexity.** Most of the problems shown decidable in this paper are easily shown to be at least as hard as the VAS reachability problem. In particular, this applies to unary separability of VAS reachability sets, and to regular separability of commutative closures of VAS languages. Indeed, for unary separability, it suffices to notice that a configuration  $u$  cannot reach a configuration  $v$  if, and only if, the set reachable from  $u$  can be unary separated from the singleton set  $\{v\}$ , also a VAS reachability set. When the separator exists, it can be taken to be the complement of  $\{v\}$  itself, which is unary.

While the problem of modular separability is EXPSPACE-hard, we do not know whether it is as hard as the VAS reachability problem. The hardness can be shown by reduction from the control state reachability problem in VASSes, which is EXPSPACE-hard [18]. For a VASS  $V$  and a target control state  $q$  thereof, we construct two new VASSes  $V_0$  and  $V_1$ , which are copies of  $V$  with one additional coordinate, which at the beginning is zero for  $V_0$  and one for  $V_1$ . We also add one new transition from control state  $q$ , which allows  $V_1$  to decrease the additional coordinate by one. One can easily verify that the two VASS reachability sets definable by  $V_0$  and  $V_1$  are modular separable if, and only if, the control state  $q$  is not reachable in  $V$ , which finishes the proof of EXPSPACE-hardness.

**The unarity and modularity characterization problems.** Closely related problems to separability are the modularity and unarity characterization problems: is a given section of a VAS reachability set modular, resp., unary? We focus here on the unarity problem, but the other one can be dealt in the same way. Decidability of the unarity problem would follow immediately from Theorem 8, if sections of VAS reachability sets were (effectively) closed under complement. This is however not the case. Indeed, if the complement of a VAS reachability set is a section of another VAS reachability set, then both sets are necessarily a section of a Presburger invariant [16], hence semilinear. But we know that VAS reachability sets can be non-semilinear, and thus they are not closed under complement. However, the unarity problem can be shown to be decidable directly, at least for VAS reachability sets, by using the following two facts: first, it is decidable if a given VAS reachability set  $U$  is semilinear (see the unpublished works [7, 14]); second, when a VAS reachability set is semilinear, a concrete representation thereof as a semilinear set is effectively computable [15]. Indeed, if a given  $U$  is not semilinear, it is not unary either; otherwise, compute a semilinear representation, and check if it is unary. The latter can be checked directly, or can be reduced to unary separability of semilinear sets.



**Acknowledgements.** We thank Maria Donten-Bury for providing us elegant proofs of Lemmas 12 and 21, and Jerome Leroux for pointing out to us the references [7, 14, 15, 15].

---

## References

- 1 Christian Choffrut and Serge Grigorieff. Separability of rational relations in  $A^* \times \mathbb{N}^m$  by recognizable relations is decidable. *Inf. Process. Lett.*, 99(1):27–32, 2006.
- 2 Lorenzo Clemente, Wojciech Czerwiński, Sławomir Lasota, and Charles Paperman. Separability of reachability sets of vector addition systems, 2016. URL: <https://arxiv.org/abs/1609.00214>.
- 3 Wojciech Czerwiński, Wim Martens, and Tomáš Masopust. Efficient separability of regular languages by subsequences and suffixes. In *Proc. of ICALP'13*, pages 150–161, 2013.
- 4 Wojciech Czerwiński, Wim Martens, Larijn van Rooijen, and Marc Zeitoun. A note on decidable separability by piecewise testable languages. In *Proc. of FCT'15*, pages 173–185, 2015.
- 5 L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with  $n$  distinct prime factors. *American Journal of Mathematics*, 35((4)):413–422, 1913.
- 6 Jean Goubault-Larrecq and Sylvain Schmitz. Deciding piecewise testable separability for regular tree languages. In *Proc. of ICALP'16*, pages 97:1–97:15, 2016. doi:10.4230/LIPIcs.ICALP.2016.97.
- 7 D. Hauschildt. *Semilinearity of the reachability set is decidable for Petri nets*. PhD thesis, University of Hamburg, 1990.
- 8 G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Mathematical Society*, 3((2)):326–336, 1952.
- 9 John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comput. Sci.*, 8:135–159, 1979.
- 10 Thomas W. Hungerford. *Algebra*, volume 73 of *Graduate Texts in Mathematics*. Springer, 1974.
- 11 Harry B. Hunt III. On the decidability of grammar problems. *Journal of the ACM*, 29(2):429–447, 1982.
- 12 Petr Jančar. Decidability of a temporal logic problem for Petri nets. *Theor. Comput. Sci.*, 74(1):71–93, 1990.
- 13 Eryk Kopczyński. Invisible pushdown languages. In *Proc. of LICS'16*, pages 867–872, New York, NY, USA, 2016. ACM. doi:10.1145/2933575.2933579.
- 14 J. L. Lambert. Vector addition systems and semi-linearity. *SIAM J. Comp.*, 1994. Accepted for publication.
- 15 J. Leroux. Presburger vector addition systems. In *Proc. of LICS'13*, pages 23–32, 2013.
- 16 Jérôme Leroux. The general vector addition system reachability problem by Presburger inductive invariants. In *Proc. of LICS'09*, pages 4–13, 2009.
- 17 Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *Proc. of LICS'15*, pages 56–67, 2015.
- 18 Richard J. Lipton. The reachability problem requires exponential space. Technical report, Yale University, 1976.
- 19 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. In *Proc. of STOC'81*, pages 238–246, 1981.
- 20 Thomas Place, Larijn van Rooijen, and Marc Zeitoun. Separating regular languages by locally testable and locally threshold testable languages. In *Proc. of FSTTCS'13*, pages 363–375, 2013.
- 21 Thomas Place, Larijn van Rooijen, and Marc Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In *Proc. of MFCS'13*, pages 729–740, 2013.

## 24:14 Separability of Reachability Sets of Vector Addition Systems

- 22 Thomas Place and Marc Zeitoun. Going higher in the first-order quantifier alternation hierarchy on words. In *Proc. of ICALP'14*, pages 342–353, 2014.
- 23 Thomas Place and Marc Zeitoun. Separating regular languages with first-order logic. *Logical Methods in Computer Science*, 12(1), 2016.
- 24 Thomas G. Szymanski and John H. Williams. Noncanonical extensions of bottom-up parsing techniques. *SIAM Journal on Computing*, 5(2):231–250, 1976.

# Counting Edge-Injective Homomorphisms and Matchings on Restricted Graph Classes<sup>\*†</sup>

Radu Curticapean<sup>1</sup>, Holger Dell<sup>2</sup>, and Marc Roth<sup>3</sup>

- 1 Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary  
radu.curticapean@gmail.com
- 2 Saarland University, Saarbrücken, Germany; and  
Cluster of Excellence “Multimodal Computing and Interaction” (MMCI),  
Saarbrücken, Germany  
hdell@mmci.uni-saarland.de
- 3 Saarland University, Saarbrücken, Germany; and  
Cluster of Excellence “Multimodal Computing and Interaction” (MMCI),  
Saarbrücken, Germany  
mroth@mmci.uni-saarland.de

---

## Abstract

---

We consider the parameterized problem of counting all matchings with exactly  $k$  edges in a given input graph  $G$ . This problem is  $\#W[1]$ -hard (Curticapean, ICALP 2013), so it is unlikely to admit  $f(k) \cdot n^{O(1)}$  time algorithms. We show that  $\#W[1]$ -hardness persists even when the input graph  $G$  comes from restricted graph classes, such as line graphs and bipartite graphs of arbitrary constant girth and maximum degree two on one side.

To prove the result for line graphs, we observe that  $k$ -matchings in line graphs can be equivalently viewed as edge-injective homomorphisms from the disjoint union of  $k$  paths of length two into (arbitrary) host graphs. Here, a homomorphism from  $H$  to  $G$  is *edge-injective* if it maps any two distinct edges of  $H$  to distinct edges in  $G$ . We show that edge-injective homomorphisms from a pattern graph  $H$  can be counted in polynomial time if  $H$  has bounded vertex-cover number after removing isolated edges. For hereditary classes  $\mathcal{H}$  of pattern graphs, we obtain a full complexity dichotomy theorem by proving that counting edge-injective homomorphisms, restricted to patterns from  $\mathcal{H}$ , is  $\#W[1]$ -hard if no such bound exists.

Our proofs rely on an edge-colored variant of Holant problems and a delicate interpolation argument; both may be of independent interest.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Counting Problems, G.2.2 Graph Theory

**Keywords and phrases** matchings, homomorphisms, line graphs, counting complexity, parameterized complexity

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.25

## 1 Introduction

Since Valiant’s seminal  $\#P$ -hardness result for the permanent [35], the complexity theory of counting problems has advanced to a classical subfield of computational complexity. As

---

\* A full version of the paper is available at <http://arxiv.org/abs/1702.05447>.

† This work was done while the authors were visiting the Simons Institute for the Theory of Computing. Radu Curticapean is supported by ERC Starting Grant PARAMTIGHT (No. 280152)



it turned out that many interesting counting problems are  $\#P$ -hard, various *relaxations* of the original problems were introduced, giving rise to approximate [26], modular [3], and subexponential counting [17], with additional restrictions on the input classes [25, 38].

In this paper, we focus on a recent relaxation of hard counting problems by studying their *parameterized* complexity [18]. In this paradigm, the input to a given counting problem comes with a parameter  $k \in \mathbb{N}$ , and we ask whether the problem is *fixed-parameter tractable (FPT)*: That is, can it be solved in time  $f(k) \cdot \text{poly}(|x|)$  for some computable function  $f$  that may grow super-polynomially? For instance, the  $\#P$ -complete problem of counting vertex-covers of size  $k$  in an  $n$ -vertex graph can be solved in time  $2^k \cdot \text{poly}(n)$  (and even faster) and is hence FPT [18]. For other parameterized problems however, such as counting cliques, cycles, paths, or matchings of size  $k$  in  $n$ -vertex graphs, the best known algorithms run in time  $n^{O(k)}$ , and FPT-algorithms are not believed to exist. To substantiate this belief, Flum and Grohe [18] introduced the class  $\#W[1]$  and identified the problem of counting  $k$ -cliques to be complete for  $\#W[1]$  under parameterized reductions. Hence this problem is not FPT, unless the classes FPT and  $\#W[1]$  coincide, which is considered unlikely. Subsequently,  $\#W[1]$ -completeness was also shown for counting  $k$ -cycles and  $k$ -paths [18], and later on for counting  $k$ -matchings [11, 13]. Interestingly, the decision versions of these last three problems are in fact FPT [1] (or even polynomial-time solvable in the case of matchings).

As it turns out, the problem of counting  $k$ -matchings plays a central role in parameterized counting. This is partially due to its obvious similarity to Valiant's classical problem of counting perfect matchings. More importantly however,  $k$ -matchings represent an important reduction source to prove the hardness of other problems. For example, they constitute the bottleneck problem for counting general small subgraph patterns: Given a graph class  $\mathcal{H}$ , we can define a problem  $\#Sub(\mathcal{H})$  that asks, given a pattern graph  $H \in \mathcal{H}$  and a host graph  $G$ , to count the occurrences of  $H$  as a subgraph in  $G$ . The problem  $\#Sub(\mathcal{H})$  can be solved in polynomial time if the graphs in  $\mathcal{H}$  have a constant upper bound on the size of their matchings, whereas classes  $\mathcal{H}$  with matchings of unbounded size make the problem  $\#W[1]$ -complete [13]. This shows in particular that counting  $k$ -matchings is the minimal hard case for  $\#Sub(\mathcal{H})$ .

In this paper, we proceed from the  $\#W[1]$ -hardness of counting  $k$ -matchings in two directions: First, we strengthen this particular hardness result by showing that counting  $k$ -matchings remains  $\#W[1]$ -complete even on natural *restricted* graph classes, such as line graphs and bipartite graphs where one side has maximum degree 2. As an instrument in our proofs, we introduce the notion of *edge-injective homomorphisms*, which interpolates between the classical notions of homomorphisms and (subgraph) embeddings. In the second part of the paper, we study the parameterized complexity of counting edge-injective homomorphisms as a topic in itself. The proofs of lemmas, claims and theorems marked with  $\star$  appear in the full version of this paper, which can be found at <http://arxiv.org/abs/1702.05447>.

## 1.1 Counting matchings in restricted graph classes

In non-parameterized counting complexity, restrictions of hard problems to planar and bounded-degree graphs were studied extensively: We can count *perfect* matchings on planar graphs in polynomial time by the FKT method [33, 27], and several dichotomies show which counting versions of constraint satisfaction problems become easy on planar graphs [6, 2].

For the particular problem of counting (not necessarily perfect) matchings, a line of research [25, 15, 34] culminated in the work of Xia et al. [38] who showed that the problem remains  $\#P$ -hard even on planar bipartite graphs whose left and right side have maximum degree 2 and 3, respectively. In the parameterized setting, counting  $k$ -matchings is FPT

in planar or bounded-degree graphs [20], which rules out a parameterized analogue of the hardness result by Xia et al. [38]. It was however shown that counting  $k$ -matchings remains  $\#W[1]$ -complete on bipartite graphs [13], which was essential for the subsequent reductions to the general subgraph counting problem. In the first part of the paper, we find additional restricted graph classes on which counting  $k$ -matchings remains  $\#W[1]$ -complete.

### 1.1.1 Restricted bipartite graphs of high girth

In [13], the  $\#W[1]$ -completeness of counting  $k$ -matchings in bipartite graphs was actually shown for an edge-colorful variant where the edges of the bipartite graph are (not necessarily properly) colored with  $k$  colors and we wish to count  $k$ -matchings that pick exactly one edge from each color. This variant can be reduced to the uncolored one via inclusion–exclusion.

In this paper, we strengthen the  $\#W[1]$ -hardness result for counting edge-colorful  $k$ -matchings in bipartite graphs  $G$  and show that we may restrict one side of  $G$  to have maximum degree two. We may additionally assume any constant lower bound on the *girth* of  $G$ , that is, the length of the shortest cycle in  $G$ . For counting (edge-colorful)  $k$ -matchings, it is known that an algorithm with running time  $f(k) \cdot n^{o(k/\log k)}$  for any computable function  $f$  would refute the counting exponential-time hypothesis  $\#ETH$  [17]. That is, if such an algorithm existed, we could count satisfying assignments to 3-CNF formulas on  $n$  variables in time  $2^{o(n)}$ . Our result establishes the same lower bound in the restricted case.

► **Theorem 1** ( $\star$ ). *For every  $c \in \mathbb{N}$ , counting (edge-colorful or uncolored)  $k$ -matchings is  $\#W[1]$ -complete, even for bipartite graphs of girth at least  $c$  whose right side vertices have degree at most two. Furthermore, unless  $\#ETH$  fails, neither of these problems has an  $f(k) \cdot n^{o(k/\log k)}$ -time algorithm, for any computable function  $f$ .*

We sketch the proof in §3 by extending the so-called Holant problems [36, 4] to an edge-colored variant that proves to be useful for parameterized counting problems. In classical Holant problems, we are given as input a graph  $G = (V, E)$  with a signature  $f_v$  at each vertex  $v \in V$ . Here,  $f_v$  is a function  $f_v : \{0, 1\}^{I(v)} \rightarrow \mathbb{Z}$ , where  $\{0, 1\}^{I(v)}$  is the set of binary assignments to the edges incident with  $v$ . The problem is to compute  $\text{Holant}(G)$ , a sum over all binary assignments  $x \in \{0, 1\}^E$ , where each assignment  $x$  is weighted by  $\prod_{v \in V} f_v(x)$ .

In our edge-colored setting, the graph  $G$  is edge-colored and  $\text{Holant}(G)$  ranges only over assignments that pick exactly one edge from each color. We apply the recent technique of combined signatures [14] in this setting, an approach that is also implicit in [13]. This gives a reduction from counting edge-colorful  $k$ -matchings in general graphs to  $2^k$  instances of the restricted bipartite case. Previously, combined signatures were used only for problems with structural parameterizations, such as counting perfect matchings in graphs whose genus or apex number is bounded [14]. Our edge-colorful approach allows us to apply them also when the parameter is the solution size  $k$ .

### 1.1.2 Line graphs

Building upon Theorem 1, we then prove that counting  $k$ -matchings is  $\#W[1]$ -complete even when the input graph is a line graph. We also obtain a lower bound under  $\#ETH$ .

► **Theorem 2.** *The problem of counting  $k$ -matchings is  $\#W[1]$ -complete, even when restricted to line graphs. Furthermore, unless  $\#ETH$  fails, this problem does not have an  $f(k) \cdot n^{o(k/\log k)}$  time algorithm, for any computable function  $f$ .*

Line graphs are claw-free, that is, they exclude  $K_{1,3}$  as induced subgraphs. In fact, the class of line graphs can be characterized by a set  $S$  of nine (for large graphs seven) minimal subgraphs such that  $G$  is a line graph if and only if  $G$  contains none of the graphs from  $S$  as an induced subgraph [24, 37]. Line graphs can be recognized in linear time [28], and several classical NP-complete problems are polynomial-time solvable on line graphs, such as finding a maximum independent set [32], a maximum cut [23], or a maximum clique [30]. In contrast, Theorem 2 shows that counting  $k$ -matchings remains  $\#W[1]$ -hard on line graphs.

To prove Theorem 2, one might be tempted to first prove hardness of counting edge-colorful  $k$ -matchings in line graphs, and then reduce this problem via inclusion–exclusion to the uncolored case. This approach however fails: While the colored problem is easily shown to be  $\#W[1]$ -complete (even on complete graphs), we cannot use inclusion–exclusion to subsequently reduce to counting uncolored matchings, since doing so would lead to graphs that are not necessarily line graphs. Hence we do not know how to prove Theorem 2 in the framework of edge-colorful Holant problems directly, as we do for Theorem 1.

Instead, we prove Theorem 2 in §4 by means of a delicate interpolation argument, most similar to techniques used in the first hardness proof for uncolored  $k$ -matchings [11]. Using a simple gadget and  $k$ -matchings in line graphs as the oracle for our reduction, we generate a linear system of equations such that one of the unknowns is the number of  $k$ -matchings in a general input graph  $G$ , which is  $\#W[1]$ -complete to compute. The system turns out not to have full rank, but a careful analysis shows that the unknown we are interested in can still be uniquely determined in polynomial time.

### Perfect matchings in (perfect) line graphs

Completing the picture, we show that the non-parameterized problem of counting *perfect* matchings also remains  $\#P$ -hard on line graphs. This holds even for line graphs of bipartite graphs, which are known to be perfect, and which play an important role in the proof of the strong perfect graph theorem by Chudnovsky, Seymour, and Robertson [10].

► **Theorem 3** (\*). *The problem of counting perfect matchings is  $\#P$ -complete even for graphs that have maximum degree 4 and are line graphs of bipartite graphs. On the other hand, the problem is polynomial-time solvable on 3-regular line graphs.*

To prove this theorem, we invoke a dichotomy theorem for Holant problems by Cai, Lu, and Xia [7]: We show that the positive case of Theorem 3 can be reduced to a polynomial-time solvable Holant problem, while hardness in the negative case of Theorem 3 follows by reduction from a  $\#P$ -complete Holant problem. Due to space limitations the proof is deferred to the journal version of this paper.

## 1.2 Counting edge-injective homomorphisms

To prove Theorem 2, we actually prove the equivalent statement that counting edge-injective homomorphisms from the graph  $k \cdot P_2$  to host graphs  $G$  is  $\#W[1]$ -complete. Here, we write  $k \cdot P_2$  for the graph consisting of  $k$  disjoint copies of the path  $P_2$  with two edges. A homomorphism  $f$  from  $H$  to  $G$  is *edge-injective* if, for any distinct (but not necessarily disjoint) edges  $e = uv$  and  $e' = u'v'$  of  $H$ , the edges  $f(u)f(v)$  and  $f(u')f(v')$  in  $G$  are distinct (but not necessarily disjoint). The number of edge-injective homomorphisms from  $k \cdot P_2$  to  $G$  is easily seen to be equal to the number of  $k$ -matchings in  $L(G)$ , up to a factor of  $k! \cdot 2^k$ , which is the size of the automorphism group of a  $k$ -matching.

Starting from their application in the proof of Theorem 2, we observe that edge-injective homomorphisms are an interesting concept on its own, since they constitute a natural interpolation between homomorphisms and subgraph embeddings (which are vertex-injective homomorphisms). To study the complexity of counting edge-injective homomorphisms from general patterns, we define the problems  $\#\text{Hom}^*(\mathcal{H})$  for fixed graph classes  $\mathcal{H}$ : Given graphs  $H \in \mathcal{H}$  and  $G$ , the problem is to count the edge-injective homomorphisms from  $H$  to  $G$ . Similar frameworks exist for counting subgraphs [13], counting/deciding colorful subgraphs [13, 31, 22], counting/deciding induced subgraphs [9], and counting/deciding (not necessarily edge-injective) homomorphisms [21, 16]. In all of these cases, precise dichotomies are known for the parameterized complexity of the problem when the pattern is chosen from a fixed class  $\mathcal{H}$  and the parameter is  $|V(H)|$ . For instance, homomorphisms from  $\mathcal{H}$  can be counted in polynomial time if  $\mathcal{H}$  has bounded treewidth, and the problem is  $\#\text{W}[1]$ -complete otherwise [16]. A similar statement holds for the decision version of this problem, but here only the cores of the graphs in  $\mathcal{H}$  need to have bounded treewidth [21].

Our main outcome is a similar result for counting edge-injective homomorphisms: To state it, let the *weak vertex-cover number* of a graph  $G$  be defined as the size of the minimum vertex-cover in the graph obtained from  $G$  by deleting all *isolated edges*, that is, connected components with two vertices. Furthermore, a graph class  $\mathcal{H}$  is *hereditary* if  $H \in \mathcal{H}$  implies  $F \in \mathcal{H}$  for all induced subgraphs  $F$  of  $H$ .

► **Theorem 4** ( $\star$ ). *Let  $\mathcal{H}$  be any class of graphs. The problem  $\#\text{Hom}^*(\mathcal{H})$  can be solved in polynomial time if there is a constant  $c \in \mathbb{N}$  such that the weak vertex-cover number of all graphs in  $\mathcal{H}$  is bounded by  $c$ . If no such constant exists and  $\mathcal{H}$  additionally is hereditary, then  $\#\text{Hom}^*(\mathcal{H})$  is  $\#\text{W}[1]$ -complete.*

To prove this theorem in §5, we first adapt an algorithm for counting subgraphs of bounded vertex-cover number [13] to the setting of edge-injective homomorphisms. Then we use a Ramsey argument to show that any graph class with unbounded weak vertex-cover number contains one of six hard classes as induced subgraphs. This gives a full dichotomy for the complexity of  $\#\text{Hom}^*(\mathcal{H})$  on hereditary graph classes  $\mathcal{H}$ , however leaving out several interesting non-hereditary classes such as paths, cycles, and  $P_c$ -packings. Here, a  $P_c$ -packing for  $c \in \mathbb{N}$  is a disjoint union of paths  $P_c$ , that is, paths consisting of  $c$  edges. We handle these specific classes individually.

► **Theorem 5** ( $\star$ ). *The problem  $\#\text{Hom}^*(\mathcal{H})$  is  $\#\text{W}[1]$ -complete if  $\mathcal{H}$  is the class of paths, the class of cycles, or the class of  $P_c$ -packings, for any  $c \geq 2$ .*

We conclude this introduction with a possible future application of edge-injective homomorphisms. A wide open problem in parameterized complexity lies in classifying the subgraph patterns whose *existence* is easy to decide. The problem is known to be FPT on patterns of bounded treewidth [1], and it seems reasonable to believe that all classes  $\mathcal{H}$  of unbounded treewidth are  $\text{W}[1]$ -hard. However, even  $\text{W}[1]$ -hardness for the class of complete bipartite graphs was only shown recently in a major breakthrough [29]. On the other hand, the complexity is much better understood for deciding the existence of *homomorphisms* from a pattern class  $\mathcal{H}$ : As stated above, the treewidth of the cores is the criterion for the complexity dichotomy [21]. Since subgraph embeddings are vertex-injective homomorphisms, the notion of edge-injective homomorphisms interpolates between the solved case of homomorphisms and the unsolved case of subgraphs. In light of this fact, we also consider our results on edge-injective homomorphisms as an initial investigation of a potential avenue towards a dichotomy for deciding subgraph patterns.

## 2 Preliminaries

A *parameterized counting problem* is a function  $\Pi : \{0, 1\}^* \rightarrow \mathbb{N}$  that is endowed with a computable *parameterization*  $\kappa : \{0, 1\}^* \rightarrow \mathbb{N}$ ; it is *fixed-parameter tractable (FPT)* if there is a computable function  $f : \mathbb{N} \rightarrow \mathbb{N}$  and an  $f(k) \cdot \text{poly}(n)$ -time algorithm to compute  $\Pi(x)$ , where  $n = |x|$  and  $k = \kappa(x)$ .

An *fpt Turing reduction* is a Turing reduction from a problem  $(\Pi, \kappa)$  to a problem  $(\Pi', \kappa')$ , such that the reduction runs in  $f(k) \cdot \text{poly}(n)$ -time and each query  $y$  to the oracle satisfies  $\kappa'(y) \leq g(k)$ . Here, both  $f$  and  $g$  are computable functions. A problem is  $\#W[1]$ -hard if there is an fpt Turing reduction from the problem of counting the cliques of size  $k$  in a given graph; since it is believed that the latter does not have an FPT-algorithm,  $\#W[1]$ -hardness is a strong indicator that a problem is not FPT. For more details, see [19].

The *counting exponential-time hypothesis (#ETH)* is the claim that there exists a constant  $\epsilon > 0$  for which there is no  $2^{\epsilon n}$ -time algorithm to compute the number of satisfying assignments for an  $n$ -variable 3-CNF formula. For the counting  $k$ -cliques problem,  $\#ETH$  implies that there is no  $f(k) \cdot n^{o(k)}$ -time algorithm [8]. Whenever an fpt Turing reduction from counting  $k$ -cliques (or any source problem) to another parameterized counting problem increases the parameter  $k$  by at most a constant factor, then the same running time lower bound under  $\#ETH$  holds for the target problem as well.

Let  $H$  and  $G$  be graphs. A function  $\varphi : V(H) \rightarrow V(G)$  is a *homomorphism* from  $H$  to  $G$  if  $\varphi(e) \in E(G)$  holds for all  $e \in E(H)$ , where  $\varphi(\{u, v\}) = \{\varphi(u), \varphi(v)\}$ . The set of all homomorphisms from  $H$  to  $G$  is denoted by  $\text{Hom}(H, G)$ . A homomorphism  $\varphi \in \text{Hom}(H, G)$  is called *edge-injective* if all  $e, f \in E(H)$  with  $e \neq f$  satisfy  $\varphi(e) \neq \varphi(f)$ . We denote the set of all edge-injective homomorphisms from  $H$  to  $G$  by  $\text{Hom}^*(H, G)$ . A homomorphism  $\varphi \in \text{Hom}(H, G)$  is an *embedding* of  $H$  in  $G$  if it is a (vertex-)injective homomorphism from  $H$  to  $G$ . The set of all embeddings from  $H$  to  $G$  is denoted by  $\text{Emb}(H, G)$ .

For a class  $\mathcal{H}$  of graphs, let  $\#\text{Hom}^*(\mathcal{H})$  denote the following computational problem: Given  $H \in \mathcal{H}$  and a simple graph  $G$ , compute the number  $\#\text{Hom}^*(H, G)$ , parameterized by  $|V(H)|$ . The problems  $\#\text{Hom}(\mathcal{H})$  and  $\#\text{Emb}(\mathcal{H})$  are defined analogously.

The *line graph*  $L(G)$  of a simple graph  $G$  is the graph whose vertex set satisfies  $V(L(G)) = E(G)$  such that  $e, f \in E(G)$  with  $e \neq f$  are adjacent in  $L(G)$  if and only if the edges  $e$  and  $f$  are incident to the same vertex in  $G$ . A line graph is called *line-perfect* if it is the line graph of a bipartite graph.

## 3 Matchings in restricted bipartite graphs

In this section, we prove Theorem 1. Our arguments make heavy use of  *$k$ -edge-colored graphs*, for  $k \in \mathbb{N}$ , which are graphs  $G$  with a (not necessarily proper) edge-coloring  $c : E(G) \rightarrow [k]$ . A matching in  $G$  is *colorful* if it contains exactly one edge from each color. We let  $\#\text{ColMatch}(G)$  be the number of such matchings and  $\#\text{ColMatch}$  be the corresponding computational problem; this problem is  $\#W[1]$ -hard by the following theorem.

► **Theorem 6** ([13], Theorem 1.2). *The problem  $\#\text{ColMatch}$  is  $\#W[1]$ -complete. Unless  $\#ETH$  fails, it cannot be solved in time  $f(k) \cdot n^{o(k/\log k)}$  for any computable  $f$ .*

A straightforward application of the inclusion–exclusion principle reduces the edge-colorful version to the uncolored one (see, e.g., [12, Lemma 1.34] or [13, Lemma 2.7]).

► **Lemma 7.** *There is an fpt Turing reduction from  $\#\text{ColMatch}$  for  $k$ -edge-colored graphs to the problem of counting  $k$ -matchings in uncolored subgraphs of  $G$ ; the reduction makes at most  $2^k$  queries, each query is a subgraph of  $G$ , and the parameter of each query is  $k$ .*



### 3.1 Colorful Holant problems

We first adapt Holant problems to an edge-colorful setting by introducing *edge-colored signature graphs* and *colorful Holant problems*. In the uncolored setting, the notion of a “Holant” was introduced by Valiant [36] and later developed to a general theory of Holant problems by Cai, Lu, Xia, and various other authors [4, 5]. In Section 3.2, we will use colorful Holants to prove Theorem 1 by a reduction from  $\#\text{ColMatch}$ . A more general exposition of this material appears in the first author’s PhD thesis [12, Chapters 2 and 5.2].

► **Definition 8.** For a graph  $G$ , we denote the edges incident with a given vertex  $v \in V(G)$  by  $I(v)$ . For  $k \in \mathbb{N}$ , a *k-edge-colored signature graph* is a  $k$ -edge-colored graph  $\Omega$  that has a *signature*  $f_v : \{0, 1\}^{I(v)} \rightarrow \mathbb{Q}$  associated with each vertex  $v \in V(\Omega)$ . The graph underlying  $\Omega$  may feature parallel edges.

Given such an  $\Omega$ , denote its color classes by  $E_1, \dots, E_k \subseteq E(\Omega)$ . An assignment  $x \in \{0, 1\}^{E(\Omega)}$  is *colorful* if, for each  $i \in [k]$ , there is exactly one edge  $e \in E_i$  with  $x(e) = 1$ . Given a set  $S \subseteq E(\Omega)$ , we write  $x|_S$  for the restriction of  $x$  to  $S$ , which is the unique assignment in  $\{0, 1\}^S$  that agrees with  $x$  on  $S$ . We then define  $\text{ColHolant}(\Omega)$  as the sum

$$\text{ColHolant}(\Omega) = \sum_{\substack{x \in \{0, 1\}^{E(\Omega)} \\ \text{colorful}}} \prod_{v \in V(\Omega)} f_v(x|_{I(v)}).$$

If all signatures in  $\Omega$  map to  $\{0, 1\}$ , then  $\text{ColHolant}(\Omega)$  simply counts those edge-colorful assignments  $x$  with  $f_v(x|_{I(v)}) = 1$  for all  $v \in V(\Omega)$ . In the following, we will use this simple fact to rephrase  $\#\text{ColMatch}$  as  $\text{ColHolant}(\Omega)$  for an edge-colored signature graph  $\Omega$ .

For assignments  $x \in \{0, 1\}^*$ , write  $\text{hw}(x)$  for the Hamming weight of  $x$ . For a statement  $\varphi$ , let  $[\varphi]$  be defined to be 1 if  $\varphi$  holds and 0 otherwise.

► **Fact 9.** Let  $k \in \mathbb{N}$  and let  $G$  be a  $k$ -edge-colored graph. Define the *k-edge-colored signature graph*  $\Omega = \Omega(G)$  by associating with each vertex  $v \in V(G)$  the signature  $\text{hw}_{\leq 1} : \{0, 1\}^{I(v)} \rightarrow \{0, 1\}$ ; for any  $x \in \{0, 1\}^*$ , this signature is defined as  $\text{hw}_{\leq 1}(x) = [\text{hw}(x) \leq 1]$ . Then we can verify that  $\text{ColHolant}(\Omega) = \#\text{ColMatch}(G)$  holds.

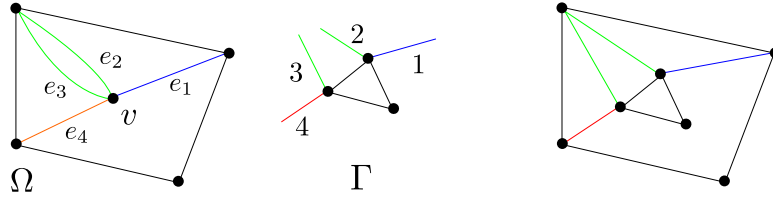
If a signature graph  $\Omega$  has a vertex  $v$  with some complicated signature  $f$  associated with it, we can sometimes simulate the effect of  $f$  by replacing  $v$  with a graph fragment that has only the signature  $\text{hw}_{\leq 1}$  associated with its vertices. Since  $\text{ColHolant}(\Omega)$  of signature graphs  $\Omega$  featuring only  $\text{hw}_{\leq 1}$  can be expressed as a number of edge-colorful matchings via Fact 9, this will allow us to reduce from  $\text{ColHolant}(\Omega)$  to  $\#\text{ColMatch}$ . The graph fragments we are looking for are formalized as *edge-colored matchgates*:

► **Definition 10.** An *edge-colored matchgate* is an edge-colored signature graph  $\Gamma$  that contains a set  $D \subseteq E(\Gamma)$  of *dangling* edges. These are edges with only one endpoint in  $V(\Gamma)$ , and we consider them to be labeled with  $1, \dots, |D|$ . Furthermore, we require the signature  $\text{hw}_{\leq 1}$  to be associated with all vertices in  $\Gamma$ . The colors on edges  $E(\Gamma) \setminus D$  will be denoted as *internal* colors.

We say that an assignment  $y \in \{0, 1\}^{E(\Gamma)}$  extends an assignment  $x \in \{0, 1\}^D$  if  $y$  agrees with  $x$  on  $D$ . The signature  $\text{ColSig}(\Gamma) : \{0, 1\}^D \rightarrow \mathbb{Q}$  of  $\Gamma$  is defined as

$$\text{ColSig}(\Gamma, x) = \sum_{\substack{y \in \{0, 1\}^{E(\Gamma)} \\ \text{colorful, extends } x}} \prod_{v \in V(\Gamma)} f_v(y|_{I(v)}).$$

If  $\Omega$  is a  $k$ -edge-colored signature graph and  $\Gamma$  is an edge-colored matchgate with internal colors disjoint from  $[k]$ , then we can *insert*  $\Gamma$  at a vertex  $v \in V(\Omega)$  as follows (see Figure 1):



■ **Figure 1** A matchgate  $\Gamma$  is inserted into a signature graph  $\Omega$  at vertex  $v$ .

First delete  $v$  from  $\Omega$ , but keep  $I(v)$  as dangling edges in  $\Omega$ . Then insert a disjoint copy of  $\Gamma$  into  $\Omega$  and identify its dangling edges with  $I(v)$ . That is, if  $e$  is a dangling edge with endpoint  $u$  in  $\Omega$  and  $e$  is identified with a dangling edge of the same color with endpoint  $v$  in  $\Gamma$ , then we consider  $e$  as an edge  $uv$  in the resulting graph.<sup>1</sup>

► **Remark.** When inserting  $\Gamma$  into a signature graph  $\Omega$ , we implicitly assume that the edge-colors of dangling edges are a subset of the edge-colors in  $\Omega$ . Furthermore, note that the insertion of matchgates can result in multigraphs.

A simple calculation shows that inserting a matchgate  $\Gamma$  at a vertex  $v$  with signature  $f_v$  preserves  $\text{ColHolant}(\Omega)$ , provided that  $\text{ColSig}(\Gamma) = f_v$ . Applying this insertion operation repeatedly, we obtain the following fact, as proved in Fact 2.17 and Lemma 5.16 of [12].

► **Fact 11.** *Let  $\Omega$  be a  $k$ -edge-colored signature graph such that each  $v \in V(\Omega)$  is associated with some signature  $f_v$ . If there is a matchgate  $\Gamma_v$  with  $\text{ColSig}(\Gamma_v) = f_v$  for every vertex  $v$ , then we can efficiently construct an edge-colored graph  $G$  on  $O(\sum_v |V(\Gamma_v)| + \sum_v |E(\Gamma_v)|)$  vertices and edges such that  $\text{ColHolant}(\Omega) = \#\text{ColMatch}(G)$ .*

In some cases, Fact 11 may not be applicable, since the involved signatures cannot be realized by matchgates. For such cases, Curticapean and Xia [14] define *combined signatures*: Rather than realizing a given signature  $f$  via matchgates, we may be able to express  $f$  as a linear combination of  $t \in \mathbb{N}$  signatures that do admit matchgates. If there are  $s \in \mathbb{N}$  occurrences of such signatures in  $\Omega$ , then we can compute  $\text{ColHolant}(\Omega)$  as a linear combination of  $t^s$  colorful Holants, where all involved signatures can be realized by matchgates.

► **Lemma 12** ( $\star$ ). *Let  $\Omega$  be a  $k$ -colored signature graph. Let  $s, t \in \mathbb{N}$  and let  $w_1, \dots, w_s$  be distinct vertices of  $\Omega$  such that the following holds: For all  $\kappa \in [s]$ , the signature  $f_\kappa$  at  $w_\kappa$  admits coefficients  $c_{\kappa,1}, \dots, c_{\kappa,t} \in \mathbb{Q}$  and signatures  $g_{\kappa,1}, \dots, g_{\kappa,t}$  such that  $f_\kappa = \sum_{i=1}^t c_{\kappa,i} \cdot g_{\kappa,i}$  holds. Here, the linear combination is to be understood point-wise.*

*Given a tuple  $\theta \in [t]^s$ , let  $\Omega_\theta$  be the edge-colored signature graph defined by replacing, for each  $\kappa \in [s]$ , the signature  $f_\kappa$  at  $w_\kappa$  with  $g_{\kappa,\theta(\kappa)}$ . Then we have*

$$\text{ColHolant}(\Omega) = \sum_{\theta \in [t]^s} \left( \prod_{\kappa=1}^s c_{\kappa,\theta(\kappa)} \right) \cdot \text{ColHolant}(\Omega_\theta). \quad (1)$$

Lemma 12 allows us to prove hardness results under fpt Turing reductions if  $\text{ColHolant}(\Omega)$  is  $\#\text{W}[1]$ -hard to compute and the values  $\text{ColHolant}(\Omega_\theta)$  for all  $\theta$  can be computed by reductions to the target problem. This is our approach in the remainder of this section.

<sup>1</sup> We assume  $I(v)$  to be ordered as  $e_1, \dots, e_{d(v)}$  in some arbitrary way; then  $e_b$  is identified with dangling edge  $b$  for all  $b \in [d(v)]$ . This also requires  $e_b$  and dangling edge  $b$  to have the same color.

### 3.2 $k$ -Matchings in bipartite graphs

In the following, we use the techniques from Section 3.1 to prove Theorem 1. We reduce from  $\#\text{ColMatch}$ , which is  $\#\text{W}[1]$ -complete by Theorem 6. Let  $k \in \mathbb{N}$  and let  $G$  be a simple  $k$ -edge-colored graph for which we want to compute  $\#\text{ColMatch}(G)$ . We first construct a certain bipartite signature graph  $\Omega_{\text{bip}}$  with  $\text{ColHolant}(\Omega_{\text{bip}}) = \#\text{ColMatch}(G)$ .

► **Lemma 13** ( $\star$ ). *Given a  $k$ -edge-colored graph  $G$ , let  $\Omega_{\text{bip}} = \Omega_{\text{bip}}(G)$  denote the signature graph on edge-colors  $[k] \times [2]$  constructed as follows: Initially,  $\Omega_{\text{bip}}$  is  $G$ , where each vertex is associated with the signature  $\text{hw}_{\leq 1}$ . Then, for each  $i \in [k]$ , perform the following:*

1. Add a fresh vertex  $w_i$  to  $\Omega_{\text{bip}}$ .
2. For  $e \in E(G)$ , of color  $i$  and with  $e = uv$ , delete  $e$  and insert an edge  $uw_i$  of color  $(i, 1)$  and an edge  $w_iv$  of color  $(i, 2)$ . Annotate the added edges with  $\pi(uw_i) = \pi(w_iv) = e$ .
3. Note that every colorful assignment  $x \in \{0, 1\}^{I(w_i)}$  at a vertex  $w_i$  has precisely two edges  $e_1(x)$  and  $e_2(x)$  that are incident to  $w_i$  and assigned 1 by  $x$ . We associate  $w_i$  with the signature  $f_i$  that maps  $x \in \{0, 1\}^{I(w_i)}$  to  $f_i(x) = [\pi(e_1(x)) = \pi(e_2(x))]$ .

The constructed signature graph  $\Omega_{\text{bip}}$  satisfies  $\text{ColHolant}(\Omega_{\text{bip}}) = \#\text{ColMatch}(G)$ .

We now realize the signatures  $f_i$  in  $\Omega_{\text{bip}}$  by linear combinations of the signatures of edge-colored matchgates. For  $i \in [k]$ , let  $E_i(G)$  denote the edges of color  $i$  in  $G$ . Let  $m_i = |E_i(G)|$  and consider the edges in  $E_i(G)$  to be ordered in some arbitrary fixed way.

► **Lemma 14** ( $\star$ ). *Recall the definition of  $\Omega_{\text{bip}}$  from Lemma 13. For  $i \in [k]$ , let  $m = m_i$  and let  $\Gamma_{i,1}$  denote the matchgate on dangling edges  $I(w_i)$  that consists of  $2m$  vertices and is defined as follows: First, create independent sets  $a_1, \dots, a_m$  and  $b_1, \dots, b_m$ , which we call “external” vertices. Then, for all  $j \in [m]$  and all edges  $e, e' \in E(\Omega_{\text{bip}})$  of colors  $(i, 1)$  and  $(i, 2)$  with  $\pi(e) = \pi(e')$ : If  $\pi(e)$  is the  $j$ -th edge in the ordering of  $E_i(G)$ , for  $j \in \mathbb{N}$ , then attach  $e$  as dangling edge to  $a_j$  and  $e'$  as dangling edge to  $b_j$ .*

Let  $\Gamma_{i,2}$  be defined likewise, with the following addition: For all  $j \in [m]$ , add an extra vertex  $c_j$ , an edge  $a_jc_j$  of color  $(i, 3)$  and an edge  $c_jb_j$  of color  $(i, 4)$ .

Recall the signature  $f_i$  from Lemma 13. We can express  $f_i$  as a linear combination of  $\text{ColSig}(\Gamma_{i,1})$  and  $\text{ColSig}(\Gamma_{i,2})$  by  $f_i = (m^2 - 3m + 3) \cdot \text{ColSig}(\Gamma_{i,1}) - \text{ColSig}(\Gamma_{i,2})$ .

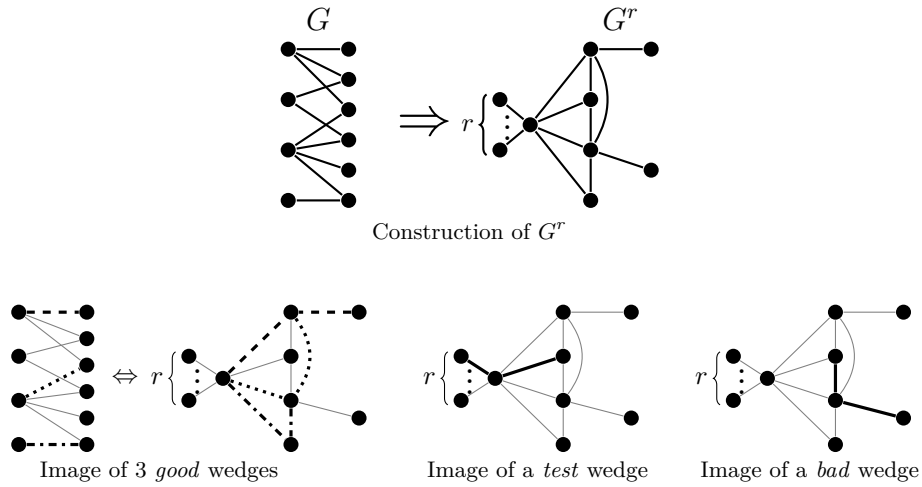
Using Lemmas 12, 13 and 14, we can now reduce counting edge-colorful matchings in graphs  $G$  to the same problem in subdivisions of  $G$ . If  $G$  is a  $k$ -colored graph and  $t \in \mathbb{N}$  is some number, then a  $t$ -subdivision of  $G$  for  $t \in \mathbb{N}$  is obtained by replacing each edge of  $G$  by a path with exactly  $t$  inner vertices. We may assign any colors to the new edges.

► **Lemma 15** ( $\star$ ). *Let  $G$  be a  $k$ -edge-colored graph on  $n$  vertices and  $m$  edges. Then we can compute  $\#\text{ColMatch}(G)$  with  $O(2^k)$  oracle calls  $\#\text{ColMatch}(G')$  for graphs  $G'$  that are subgraphs of a 3-subdivision of  $G$ . Furthermore,  $G'$  has at most  $4(n + m)$  vertices and edges and at most  $4k$  colors.*

Theorem 1 now follows easily from the hardness of  $\#\text{ColMatch}$  and repeated applications of Lemma 15. The full proof is given in the full version of this paper.

## 4 Matchings in line graphs

We now sketch the proof of Theorem 2, stating that counting  $k$ -matchings in line graphs is  $\#\text{W}[1]$ -hard. A *wedge* is any graph isomorphic to  $P_2$ , the path with two edges, and a *wedge packing*  $k \cdot P_2$  is the vertex-disjoint union of  $k$  wedges. For any graph  $G$ , we observe that the number of embeddings of a  $k$ -matching in  $L(G)$  is equal to the number of edge-injective



■ **Figure 2** Example of the construction of  $G^r$  as used in the proof of Theorem 17. The second row illustrates the correspondence between a 3-matching in  $G$  and the image of an edge-injective homomorphism from a wedge packing of size 3 such that all wedges are *good*. Furthermore we give examples for the image of a *test* wedge and a *bad* wedge.

homomorphisms from a wedge packing  $k \cdot P_2$  to  $G$ . To prove Theorem 2, we reduce from the  $k$ -matching problem in well-structured bipartite graphs to the latter problem. The following technical lemma encapsulates the delicate interpolation argument used in the reduction. For  $t \in \mathbb{N}$ , let  $(x)_t = (x) \cdot (x-1) \cdots (x-t+1)$  denote the falling factorial.

▶ **Lemma 16** ( $\star$ ). *For all  $g, b \in \mathbb{N}$ , let  $a_{g,b} \in \mathbb{Q}$  be unknowns, and for all  $r \in \mathbb{N}$ , let  $P_r(y)$  be the univariate polynomial such that*

$$P_r(y) = \sum_{k=0}^r \sum_{t=0}^k a_{t,k-t} \cdot \binom{r}{k} \cdot (y-t)_{2(r-k)}.$$

*There is a polynomial-time algorithm that, given a number  $k$  and the coefficients of  $P_r(y)$  for all  $r \in \mathbb{N}$  with  $r \leq O(k)$ , computes the numbers  $a_{t,k-t}$  for all  $t \in \{0, \dots, k\}$ .*

We then prove Theorem 2 by showing the following equivalent theorem.

▶ **Theorem 17.** *If  $\mathcal{H}$  is the class of all wedge packings, then  $\#\text{Hom}^*(\mathcal{H})$  is  $\#\text{W}[1]$ -hard. Furthermore, unless  $\#\text{ETH}$  fails, the problem cannot be solved in time  $f(k) \cdot n^{o(k/\log k)}$ .*

**Proof.** We reduce from the problem of counting  $k$ -matchings in bipartite graphs whose right-side vertices have degree  $\leq 2$  and where any two distinct left-side vertices have at most one common neighbor. For this problem, Theorem 1 for bipartite graphs with girth greater than 4 implies  $\#\text{W}[1]$ -hardness and the desired bound under  $\#\text{ETH}$ . Let  $(G, k)$  be an instance of this problem, and let  $L(G)$  and  $R(G)$  be the left and right vertex sets, respectively. For  $r \in \mathbb{N}$ , we construct a graph  $G^r$  as follows (see Figure 2):

1. Insert a vertex 0 that is adjacent to all vertices of  $L(G)$ .
2. Add  $r$  special vertices  $1, \dots, r$  as well as the edges  $01, 02, \dots, 0r$ .
3. For every vertex  $v \in R(G)$  with  $\deg(v) = 2$ , remove  $v$  and add the set  $N(v)$  as an edge to  $G^r$ . Note that  $|N(v)| = 2$ , so  $N(v)$  can indeed be considered as an edge.

Since  $G$  is a simple graph and any two distinct vertices  $u, v \in L(G)$  have at most one common neighbor in  $G$ , the graph  $G^r$  is again simple. Let  $H = H_1 \dot{\cup} \dots \dot{\cup} H_k$  be the graph that consists of  $k$  vertex-disjoint copies of  $P_2$ . For  $\varphi \in \text{Hom}^*(H, G^0)$ , we say that a wedge  $H_i$  is

- *test* if  $\varphi(H_i)$  contains two edges incident to 0,
- *good* if  $\varphi(H_i)$  contains exactly one edge incident to 0, and
- *bad* if  $\varphi(H_i)$  uses no edge incident to 0.

Let  $\alpha_{g,b}$  be the number of edge-injective homomorphisms  $\varphi \in \text{Hom}^*(H, G^0)$  for which there are 0 test wedges,  $g$  good wedges, and  $b$  bad wedges.

► **Claim 18** ( $\star$ ). *The number of  $k$ -matchings in  $G$  is equal to  $\alpha_{k,0}/(2^k \cdot k!)$ .*

We aim at determining the number  $\alpha_{k,0}$  by using an oracle for  $\#\text{Hom}^*(\mathcal{H})$ . Since we cannot directly ask the oracle to only count homomorphisms with a given number of bad and good wedges, we query the oracle multiple times and recover these numbers via a very specific form of interpolation fueled by Lemma 16. To apply the lemma, we observe the following identity.

► **Claim 19** ( $\star$ ). *Let  $k, r \in \mathbb{N}$ . Then  $\beta_k(G^r) := \#\text{Hom}^*(H, G^r)$  satisfies*

$$\beta_k(G^r) = \sum_{\substack{t, g, b \in \mathbb{N} \\ t+g+b=k}} \alpha_{g,b} \cdot \binom{k}{g+b} \cdot (n+r-g)_{2t}.$$

Note that  $\beta_k(G^r)$  is a polynomial in  $r$  of degree at most  $2k$ . Setting  $y = n+r$ , Claim 19 yields a polynomial identity that is exactly of the form required by Lemma 16, and thus we can compute the unknowns  $\alpha_{g,b}$  for all  $g, b \in \mathbb{N}$  with  $g+b \leq k$  from the polynomials  $\beta_0, \dots, \beta_{O(k)}$ . Overall, the reduction runs in polynomial time, makes at most  $O(k^2)$  queries to the oracle, and the parameter of each query is at most  $O(k)$ . This proves the  $\#\text{W}[1]$ -hardness and the lower bound under  $\#\text{ETH}$ . ◀

## 5 Edge-injective homomorphisms

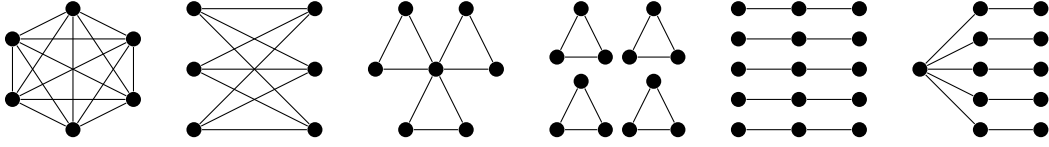
We sketch the proof of Theorem 4, our dichotomy theorem for counting edge-injective homomorphisms. Let  $H$  be a graph. Recall that a set  $S \subseteq V(H)$  is a *weak vertex-cover* if every edge  $e \in E(H)$  either has a non-empty intersection with  $S$  or  $e$  does not have any other edges incident to it. The *weak vertex-cover number* of  $G$  is the minimum size of a weak vertex-cover of  $G$ . A family of graphs  $\mathcal{H}$  has *bounded* weak vertex-cover number if this number can be uniformly bounded by a constant  $c = c(\mathcal{H})$  for all graphs  $H \in \mathcal{H}$ ; otherwise this number is *unbounded*.

### 5.1 Polynomial-time counting for bounded weak vertex-cover number

The polynomial-time cases of our dichotomy are established in the following theorem.

► **Theorem 20** ( $\star$ ). *If  $\mathcal{H}$  is a family of graphs with bounded weak vertex-cover number, then  $\#\text{Hom}^*(\mathcal{H})$  is polynomial-time computable.*

The algorithm is based on dynamic programming. Let  $H \in \mathcal{H}$  and  $G$  be the input for the algorithm. Isolated edges of  $H$  can be removed easily, as their contribution to the number of edge-injective homomorphisms admits a closed formula. The basic idea now is to guess which  $c$  vertices in  $G$  the vertex-cover of  $H$  maps to; after this part of the homomorphism is fixed, all vertices of  $H$  outside of the vertex-cover form an independent set, and they



■ **Figure 3** Example graphs from each of the six minimal graph classes that do not have bounded weak vertex-cover number according to Lemma 21:  $K_6$ ,  $K_{3,3}$ ,  $W_3$ ,  $4 \cdot K_3$ ,  $5 \cdot P_2$ , and  $SS_5$ .

can only be distinguished if their neighborhoods are distinct. Since there are at most  $2^c$  different neighborhoods, the graph  $H$  has a very simple structure, and a surprisingly technical dynamic programming algorithm achieves a running time of  $n^{O(2^c)}$ .

## 5.2 Hardness for hereditary graph classes

We now consider graph classes  $\mathcal{H}$  that do *not* have bounded weak vertex-cover number, and we prove that  $\#\text{Hom}^*(\mathcal{H})$  is  $\#\text{W}[1]$ -complete if  $\mathcal{H}$  is also hereditary. To do so, we first show that every class of unbounded weak vertex-cover number contains one of six basic graph classes (depicted in Figure 3) as induced subgraphs.

For the purposes of this paper, we say that  $W_k$  is a windmill of size  $k$  if it is a matching of size  $k$  with an additional *center vertex* adjacent to every other vertex. Moreover, the *subdivided star*  $SS_k$  is a  $k$ -matching with a center vertex that is adjacent to exactly one vertex of each edge in the matching. A *triangle packing*  $k \cdot K_3$  is the disjoint union of  $k$  triangles, a *wedge* is a path  $P_2$  that consists of two edges, and a *wedge packing*  $k \cdot P_2$  is the disjoint union of  $k$  wedges.

► **Lemma 21** ( $\star$ ). *Let us say that a class  $\mathcal{H}$  contains another class  $\mathcal{C}$  as induced subgraphs if, for every  $C \in \mathcal{C}$ , there is some  $H \in \mathcal{H}$  such that  $H$  contains  $C$  as induced subgraph. If  $\mathcal{H}$  is a class of graphs with unbounded weak vertex-cover number, then  $\mathcal{H}$  contains at least one of the following classes as induced subgraphs:*

- (i) *the class of all cliques,*
- (ii) *the class of all bicliques,*
- (iii) *the class of all subdivided stars,*
- (iv) *the class of all windmills,*
- (v) *the class of all triangle packings, or*
- (vi) *the class of all wedge packings.*

Since hereditary classes  $\mathcal{H}$  are closed under induced subgraphs, Lemma 21 guarantees that any hereditary class  $\mathcal{H}$  with unbounded weak vertex-cover number contains at least one of the six graph families defined above as an actual subset of  $\mathcal{H}$ . We prove hardness for each of these six families in the journal version of this paper; the hardness for hereditary classes  $\mathcal{H}$  and Theorem 4 then follows.

## 5.3 Hardness for some non-hereditary graph classes

The dichotomy for  $\#\text{Hom}^*(\mathcal{H})$  with hereditary graph classes  $\mathcal{H}$  leaves open some non-hereditary graph classes of interest. In the final part of the paper, we investigate  $\#\text{Hom}^*(\mathcal{H})$  for several such classes, namely those of cycles, paths, and packings of constant-length paths. It turns out that the problem of counting edge-injective homomorphisms is  $\#\text{W}[1]$ -hard in all of these cases (excluding the class of matchings, which are packings of length-1 paths).

► **Theorem 22** ( $\star$ ). For the classes  $\mathcal{C}$  and  $\mathcal{P}$  of all cycles and paths, respectively, the problems  $\#\text{Hom}^*(\mathcal{C})$  and  $\#\text{Hom}^*(\mathcal{P})$  are  $\#\text{W}[1]$ -hard. Furthermore, the problem of counting all edge-disjoint  $s$ - $t$ -walks in a given graph is  $\#\text{W}[1]$ -hard.

► **Theorem 23** ( $\star$ ). For  $c \in \mathbb{N}$ , let  $\mathcal{PP}_c$  be the class of packings of the path  $P_c$ . Then  $\#\text{Hom}^*(\mathcal{PP}_c)$  is  $\#\text{W}[1]$ -hard for  $c \geq 2$  and computable in polynomial time otherwise.

**Acknowledgments.** The authors thank Cornelius Brand and Markus Bläser for interesting discussions, and Johannes Schmitt for pointing out a proof of Lemma 16.

---

## 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 Jin-Yi Cai and Zhiguo Fu. Holographic algorithm with matchgates is universal for planar  $\#\text{CSP}$  over boolean domain. *CoRR*, abs/1603.07046, 2016. URL: <http://arxiv.org/abs/1603.07046>.
- 3 Jin-Yi Cai and Lane A. Hemachandra. On the power of parity polynomial time. *Mathematical Systems Theory*, 23(2):95–106, 1990. doi:10.1007/BF02090768.
- 4 Jin-Yi Cai and Pinyan Lu. Holographic algorithms: From art to science. In *Proceedings of the 39th ACM Symposium on Theory of Computing, STOC*, pages 401–410, 2007. doi:10.1145/1250790.1250850.
- 5 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. Holographic algorithms by Fibonacci gates and holographic reductions for hardness. In *Proc. of the 49th Annual Symposium on Foundations of Computer Science, FOCS*, pages 644–653, 2008. doi:10.1109/FOCS.2008.34.
- 6 Jin-yi Cai, Pinyan Lu, and Mingji Xia. Holographic algorithms with matchgates capture precisely tractable planar  $\#\text{CSP}$ . In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 427–436, 2010. doi:10.1109/FOCS.2010.48.
- 7 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. Computational complexity of holant problems. *SIAM Journal on Computing*, 40(4):1101–1132, 2011. doi:10.1137/100814585.
- 8 Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David W. Juedes, Iyad A. Kanj, and Ge Xia. Tight lower bounds for certain parameterized NP-hard problems. *Information and Computation*, 201(2):216–231, 2005. doi:10.1016/j.ic.2005.05.001.
- 9 Yijia Chen, Marc Thurley, and Mark Weyer. Understanding the complexity of induced subgraph isomorphisms. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, ICALP*, pages 587–596, 2008. doi:10.1007/978-3-540-70575-8\_48.
- 10 Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Annals of mathematics*, pages 51–229, 2006.
- 11 Radu Curticapean. Counting matchings of size  $k$  is  $\#\text{W}[1]$ -hard. In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming, ICALP*, pages 352–363, 2013. doi:10.1007/978-3-642-39206-1\_30.
- 12 Radu Curticapean. *The simple, little and slow things count: On parameterized counting complexity*. PhD thesis, Saarland University, August 2015.
- 13 Radu Curticapean and Dániel Marx. Complexity of counting subgraphs: Only the boundedness of the vertex-cover number counts. In *Proceedings of the 55th Annual Symposium on Foundations of Computer Science, FOCS*, pages 130–139. IEEE, 2014. doi:10.1109/FOCS.2014.22.

- 14 Radu Curticapean and Mingji Xia. Parameterizing the permanent: Genus, apices, minors, evaluation mod  $2^k$ . In *Proceedings of the 56th Annual Symposium on Foundations of Computer Science, FOCS*, pages 994–1009, 2015. doi:10.1109/FOCS.2015.65.
- 15 Paul Dagum and Michael Luby. Approximating the permanent of graphs with large factors. *Theoretical Computer Science*, 102(2):283–305, 1992. doi:10.1016/0304-3975(92)90234-7.
- 16 Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theoretical Computer Science*, 329(1-3):315–323, 2004. doi:10.1016/j.tcs.2004.08.008.
- 17 Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlen. Exponential time complexity of the permanent and the Tutte polynomial. *ACM Transactions on Algorithms*, 10(4):21, 2014. doi:10.1145/2635812.
- 18 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal of Computing*, 33(4):892–922, 2004. doi:10.1137/S0097539703427203.
- 19 Jörg Flum and Martin Grohe. *Parameterized complexity theory*. Springer, 2006.
- 20 Markus Frick. Generalized model-checking over locally tree-decomposable classes. *Theoretical Computer Science*, 37(1):157–191, 2004. doi:10.1007/s00224-003-1111-9.
- 21 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1):1, 2007. doi:10.1145/1206035.1206036.
- 22 Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable? In *Proceedings of the 33rd ACM Symposium on Theory of Computing, STOC*, pages 657–666, 2001. doi:10.1145/380752.380867.
- 23 Venkatesan Guruswami. Maximum cut on line and total graphs. *Discrete Applied Mathematics*, 92(2-3):217–221, 1999. doi:10.1016/S0166-218X(99)00056-6.
- 24 Frank Harary. *Graph theory*, 1969.
- 25 Mark Jerrum. Two-dimensional monomer-dimer systems are computationally intractable. *Journal of Statistical Physics*, 48(1-2):121–134, 1987. doi:10.1007/BF01010403.
- 26 Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM*, 51(4):671–697, 2004. doi:10.1145/1008731.1008738.
- 27 Pieter W. Kasteleyn. *Graph theory and crystal physics*. In *Graph Theory and Theoretical Physics*, pages 43–110. Academic Press, 1967.
- 28 Philippe G. H. Lehot. An optimal algorithm to detect a line graph and output its root graph. *Journal of the ACM*, 21(4):569–575, 1974. doi:10.1145/321850.321853.
- 29 Bingkai Lin. The parameterized complexity of  $k$ -biclique. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 605–615, 2015. doi:10.1137/1.9781611973730.41.
- 30 Vadim V. Lozin and Raffaele Mosca. Independent sets in extensions of  $2K_2$ -free graphs. *Discrete Applied Mathematics*, 146(1):74–80, 2005. doi:10.1016/j.dam.2004.07.006.
- 31 Kitty Meeks. The challenges of unbounded treewidth in parameterised subgraph counting problems. *Discrete Applied Mathematics*, 198:170–194, 2016. doi:10.1016/j.dam.2015.06.019.
- 32 Najiba Sbihi. Algorithme de recherche d’un stable de cardinalité maximum dans un graphe sans étoile. *Discrete Mathematics*, 29:53–76, 1980. doi:10.1016/0012-365X(90)90287-R.
- 33 Harold N. V. Temperley and Michael E. Fisher. Dimer problem in statistical mechanics – an exact result. *Philosophical Magazine*, 6(68):1478–6435, 1961.
- 34 Salil P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM Journal of Computing*, 31(2):398–427, 2001. doi:10.1137/S0097539797321602.
- 35 Leslie G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.



- 36 Leslie G. Valiant. Holographic algorithms. *SIAM Journal of Computing*, 37(5):1565–1594, 2008. doi:10.1137/070682575.
- 37 L. Šoltés. Forbidden induced subgraphs for line graphs. *Discrete Mathematics*, 132(1):391–394, 1994.
- 38 Mingji Xia, Peng Zhang, and Wenbo Zhao. Computational complexity of counting problems on 3-regular planar graphs. *Theoretical Computer Science*, 384(1):111–125, 2007. Theory and Applications of Models of Computation. doi:10.1016/j.tcs.2007.05.023.



# Robust and Adaptive Search\*

Yann Disser<sup>1</sup> and Stefan Kratsch<sup>2</sup>

- 1 TU Darmstadt, Darmstadt, Germany  
disser@mathematik.tu-darmstadt.de
- 2 University of Bonn, Bonn, Germany  
kratsch@cs.uni-bonn.de

---

## Abstract

Binary search finds a given element in a sorted array with an optimal number of  $\log n$  queries. However, binary search fails even when the array is only slightly disordered or access to its elements is subject to errors. We study the worst-case query complexity of search algorithms that are robust to imprecise queries and that adapt to perturbations of the order of the elements. We give (almost) tight results for various parameters that quantify query errors and that measure array disorder. In particular, we exhibit settings where query complexities of  $\log n + ck$ ,  $(1 + \varepsilon) \log n + ck$ , and  $\sqrt{cnk} + o(nk)$  are best-possible for parameter value  $k$ , any  $\varepsilon > 0$ , and constant  $c$ .

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** searching, robustness, adaptive algorithms, memory faults, array disorder

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.26

## 1 Introduction

Imagine a large register with  $n$  files from which you wish to extract a particular file. All files are indexed by some key and the files are sorted by key value. Not knowing the distribution of the keys, you probably use binary search since looking at  $\log n$  keys is best possible in the worst case. Unfortunately, however, other users have accessed files before you and have only returned the files to approximately the right place. As a result, the register is unsorted, but at least each file is within some small number  $k$  of positions of where it should be. How should you proceed? If you knew  $k$  and  $n$ , at what ratio of  $k$  vs.  $n$  should you resort to a linear search of the register? If you do not know  $k$ , can you still do reasonably well? What if the register was recently moved, by packing the files into boxes, but in the process the order of the boxes got mixed up, and now there are large blocks of files that are far away from their correct locations? What if you misread some of the keys? Situations like these are close to searching in a sorted register and there are plenty of parameters that measure closeness to a sorted array, e.g., maximum displacement or minimum block moves to sort, respectively persistent or temporary read errors. We give (almost) optimal algorithms for a large variety of these measures, and thereby establish for each of them exact regimes in which we can outperform a linear search of all elements, or even be almost as good as binary search.

More formally, we study the fundamental topic of comparison-based search, which is central to many algorithms and data structures [20, 24, 31]. In its most basic form, the search problem can be phrased in terms of locating an element  $e$  within a given array  $A$ . In order to search  $A$  efficiently, we need structure in the ordering of its elements: In general, we cannot hope to avoid querying all entries to find  $e$ . The most prominent example of an

---

\* A full version of the paper is available at <http://arxiv.org/abs/1702.05932>.



efficient search algorithm that exploits special structure is *binary search* for sorted arrays. Binary search is best-possible for this case. It needs only logarithmically many queries and is thus very well suited for searching extremely large collections of data. However, it heavily relies on perfect order and reliable access to the data. For large and dynamically changing collections of data, both requirements may be difficult to ensure, but it may be reasonable to assume the number of imperfections to be bounded. Accordingly, we ask: *What is the best-possible search algorithm if the data may be disordered or we cannot access it reliably? In what regime of the considered measure is it better than linear search?*

We provide (almost) tight bounds on the query complexity of searching an array  $A$  with  $n$  entries for an element  $e$  in a variety of settings. Each setting is characterized by bounding a different parameter  $k$  that quantifies the imperfections regarding either our access to array elements or regarding the overall disorder of the data. Note that one can always resort to linear search, which rules out lower bounds stronger than  $n$  comparisons.<sup>1</sup> Table 1 gives an overview of the parameters we analyze and our respective results. Qualitatively, our results can be grouped into three groups of settings leading to different query complexities, and we briefly highlight each group in the following.

The first group contains the parameters  $k_{\text{sum}}$ ,  $k_{\text{max}}$ , and  $k_{\text{inv}}$ , which quantify the summed and maximum distance of each element from its position in sorted order and the number of element pairs in the wrong relative order, respectively (detailed definitions can be found in the corresponding sections). For all of these parameters we are able to show that  $\log n + ck$  queries are necessary and sufficient, for constant  $c$ . Intuitively, this is the best complexity we can hope for: We cannot do better than  $\log(n)$  queries, and the impact of  $k$  on the query complexity is linear and can be isolated.

The second group of results is with respect to the parameters  $k_{\text{lies}}$ ,  $k_{\text{faults}}$ , as well as multiple parameters for edit distances that measure the number of element operations needed to sort  $A$ . The parameter  $k_{\text{lies}}$  limits the number of queries that yield the wrong result, and  $k_{\text{faults}}$  limits the number of array positions that yield wrong query outcomes. For bounded values of  $k_{\text{lies}}$  and  $k_{\text{faults}}$  we show that  $e$  cannot be found with  $\log n + ck$  queries using any binary-search-like algorithm.<sup>2</sup> On the other hand, we provide an algorithm that needs  $(1 + 1/c) \log n + ck$  queries, for any  $c \geq 1$ . For bounded edit distances, it is easy to see that we need  $n$  queries if  $e$  need not be at its correct position relative to sorted order, since  $e$  can be moved anywhere with just 2 edits, forcing us to scan the whole array. If we assume  $e$  to be at its correct location, we can carry over the results for  $k_{\text{lies}}$  and  $k_{\text{faults}}$  to obtain the same bounds for the edit-distance related parameters  $k_{\text{rep}}$ ,  $k_{\text{seq}}$ ,  $k_{\text{mov}}$ , and  $k_{\text{swap}}$ .

Lastly, we consider the parameter  $k_{\text{ainv}}$  that counts the number of adjacent elements that are in the wrong relative order, as well as several parameters measuring the number of block operations needed to sort  $A$ . Intuitively, these settings are much more difficult for a search algorithm, as it takes relatively small parameter values to introduce considerable disorder. For the case that  $e$  is guaranteed to be at the correct position, we show that  $\sqrt{cnk} + o(nk)$  queries are necessary and sufficient to locate  $e$ .

The algorithms for  $k_{\text{ainv}}$  and related parameters assume that the parameter value is known to the algorithm a priori. In contrast, all our other algorithms are oblivious to the parameter, in the sense that they do not require knowledge of the parameter value as long

<sup>1</sup> Accordingly, all (lower) bounds of the form  $f(n, k)$  throughout the paper are to be understood as  $\min\{f(n, k), n\}$ . A naive bound of  $n$  can easily be obtained by scanning the whole array.

<sup>2</sup> We interpret the array as a binary tree (rooted at entry  $n/2$ , with the two children  $n/4$ ,  $3n/4$ , etc.), and call an algorithm “binary-search-like” if it never queries a node (other than the root) before querying its parent.

■ **Table 1** Overview of our results, with main results in boldface.<sup>1</sup> (°: even if oblivious to parameter value; °: for all  $c \geq 1$ ; <sup>t</sup>: for tree-algorithms; <sup>e</sup>: for  $\text{pos}(e) = \text{rank}(e)$ )

parameter description	bounds	
	lower	upper
<b>Section 3</b> – number of imprecise queries		
$k_{\text{lies}}$ wrong outcomes	<b><math>\log n + ck</math></b> [Th. 3] <sup>ct</sup>	<b><math>(1 + \frac{1}{c}) \log n + (2c+2)k</math></b> [Th. 2] <sup>oc</sup>
$k_{\text{faults}}$ indices with wrong outcomes	$\log n + ck$ [Th. 3] <sup>ct</sup>	$(1 + \frac{1}{c}) \log n + (2c+2)k$ [Th. 4] <sup>oc</sup>
<b>Section 4.1</b> – displacement of elements		
$k_{\text{sum}}$ total displacement	<b><math>\log n/k + 2k + \mathcal{O}(1)</math></b> [Th. 5] <sup>o</sup>	
$k_{\text{max}}$ maximum displacement	<b><math>\log n/k + 3k + \mathcal{O}(1)</math></b> [Th. 6] <sup>o</sup>	
<b>Section 4.2</b> – number of inversions		
$k_{\text{inv}}$ all inversions	$\log n/k + 2k + \mathcal{O}(1)$ [Co. 7]	$\log n/k + 4k + \mathcal{O}(1)$ [Co. 7] <sup>o</sup>
$k_{\text{ainv}}$ adjacent inversions	<b><math>\sqrt{8nk} + o(\sqrt{nk})</math></b> [Th. 10,9] <sup>e</sup>	
<b>Section 4.3</b> – element operations needed to sort the array		
$k_{\text{rep}}$ element replacements	$\log n + ck$ [Co. 14] <sup>cte</sup>	$(1 + \frac{1}{c}) \log n + (4c+4)k$ [Th. 13] <sup>oe</sup>
$k_{\text{seq}}$ $n -  \text{max ordered subseq.} $	$\log n + ck$ [Co. 14] <sup>cte</sup>	$(1 + \frac{1}{c}) \log n + (4c+4)k$ [Th. 13] <sup>oe</sup>
$k_{\text{mov}}$ element moves	$\log n + ck$ [Co. 14] <sup>cte</sup>	$(1 + \frac{1}{c}) \log n + (4c+4)k$ [Th. 13] <sup>oe</sup>
$k_{\text{swap}}$ element swaps	$\log n + ck$ [Co. 14] <sup>cte</sup>	$(1 + \frac{1}{c}) \log n + (8c+8)k$ [Th. 13] <sup>oe</sup>
$k_{\text{aswap}}$ adj. element swaps	$\log n/k + 2k + \mathcal{O}(1)$ [Co. 15]	$\log n/k + 4k + \mathcal{O}(1)$ [Co. 15] <sup>o</sup>
<b>Section 4.4</b> – block operations needed to sort the array		
$k_{\text{bswap}}$ block swaps	<b><math>4\sqrt{nk} + o(\sqrt{nk})</math></b> [Th. 17] <sup>e</sup>	
$k_{\text{rbswap}}$ equal size block swaps	$2\sqrt{2nk} + o(\sqrt{nk})$ [Th. 18] <sup>e</sup>	$4\sqrt{nk} + o(\sqrt{nk})$ [Th. 18] <sup>e</sup>
$k_{\text{bmov}}$ block moves	<b><math>2\sqrt{2nk} + o(\sqrt{nk})</math></b> [Th. 19] <sup>e</sup>	

as the target element  $e$  is guaranteed to be present in the array. Note that if  $e$  need not be present and we have no bound on the disorder, we generally need to inspect every entry of the array in case we cannot find  $e$ . For the parameter  $k_{\text{lies}}$ , we do not even know how long we need to continue querying the same elements until we may conclude that  $e$  is not part of the array. Any of our oblivious algorithms can trade the guarantee that  $e \in A$  against knowledge of the parameter value  $k$ : Compute from  $k$  the maximum number  $m$  of queries that it would take without knowing  $k$  when  $e \in A$ . If the algorithm does not stop within  $m$  queries then it is safe to answer that  $e$  is not in  $A$ .

Overall, our results point out several parameters for which a fairly large regime of  $k$  (as a function of  $n$ ) allows search algorithms that are provably better than linear search. For example, while moving only a single element by a lot can lead to bounds of  $\Omega(n)$  on the values of several parameters, and hence trivial guarantees, moving many elements by at most  $k$  places gives  $k_{\text{max}} = k$  and yields better bounds than linear search (roughly) for  $k < \frac{n}{3}$ , and as good as binary search when  $k = \mathcal{O}(\log n)$ . Moving only few elements by an arbitrary number of spaces, in turn, still leads to good bounds via parameters such as  $k_{\text{mov}}$  or  $k_{\text{swap}}$ , as long as the target is in the correct place. Parameters such as  $k_{\text{ainv}}$  grow even more slowly, for certain types of disorder, but, on the other hand, only a small regime allows for better than trivial guarantees. While, for each individual parameter we study, there are “easily searchable” instances where the parameter becomes large and makes the corresponding bound trivial, our results often allow for good bounds by resorting to a different parameter.

## 1.1 Related Work

Our work falls into the area of *adaptive analysis of algorithms*, which aims at a fine-grained analysis of polynomial-time algorithms with respect to structural parameters of the input. An objective of this field is to find algorithms whose running-time dependence on input size and the structural parameters interpolates smoothly between known (good) bounds for special cases and the worst-case bound for general inputs. The topic of adaptive sorting, i.e., sorting arrays that are presorted in some sense, has attracted a lot of attention, see, e.g., [4, 13, 23, 28].

We now discuss results that are specific to searching in arrays. Several authors addressed the question of how much preprocessing, i.e., sorting, helps for searching, if we take into account the total time investment [8, 22, 29]. Fredman [18] gave lower bounds on searching regarding both queries and memory accesses. A classic work of Yao [32] established that the best way of storing  $n$  elements in a table such as to minimize number of queries for accessing an element is by keeping the elements sorted, which requires  $\log n$  queries, provided that the key space is large enough. Regarding searching in (partially) unordered arrays, there is a nice result of Biedl et al. [5] about insertion sort based on repeated binary searches.

Under appropriate assumptions, namely that array is sorted and its elements are drawn from a known distribution (e.g., searching for a name in a telephone book), one can do much better than binary search, since the distribution allows a good prediction of where the target should be located. In this case  $\mathcal{O}(\log \log n)$  queries suffice on average (cf. [31]); to avoid having to query the entire array, previous work suggests combinations of algorithms that perform no worse than binary search in the worst case [10, 6]. Another interesting branch of study is related to search in arrays of more complicated objects such as (long) strings [1, 17] or abstract objects with nonuniform comparison cost [19, 2].

Many papers have studied searching in the presence of different types of errors, e.g., [7, 15, 16, 25], see [11, 27] for surveys. A popular error model for searching allows for a linear number of lies [3, 7, 12, 14, 26], for which Borgstrom and Kosaraju [7] gave an  $\mathcal{O}(\log n)$  search algorithm. In contrast, we bound the number of lies separately via the parameter  $k_{\text{lies}}$ . Rivest et al. [30] gave an upper bound of  $\log n + k \log \log n + \mathcal{O}(k \log k)$  queries for this parameter. Their algorithm is based on a continuous strategy for the (equivalent) problem of finding an unknown value in  $[1, n]$ , up to a given precision, using few yes-no questions. Our algorithm (Theorem 2) uses asymptotically fewer queries if  $k_{\text{lies}} = \omega(\log n / \log \log n)$ .<sup>3</sup>

The works of Finocchi and Italiano [16] and Finocchi et al. [15] consider a parameter very similar to  $k_{\text{faults}}$ , with the additional assumption that faults may affect also the working memory of the algorithm, except for  $\mathcal{O}(1)$  “safe” memory words. Finocchi and Italiano [16] give a deterministic searching algorithm that needs  $\mathcal{O}(\log n + k^2)$  queries. Brodal et al. [9] improve this bound to  $\mathcal{O}(\log n + k)$  and Finocchi et al. [15] provide a lower bound of  $\Omega(\log n + k)$  even for randomized algorithms. Our results are incomparable as our result for parameter  $k_{\text{faults}}$  uses only  $(1 + \frac{1}{c}) \log n + (2c + 2)k$  queries, getting arbitrarily close to  $\log n + \mathcal{O}(k)$  (cf. Theorem 4), but does not consider faults in the working memory; the high level approach of balancing progress in the search with security queries is the same as in [9], but more careful counting is needed to get small constants. For parameter  $k_{\text{lies}}$  we give a simpler algorithm with  $2 \log n + 4k$  queries and using only  $\mathcal{O}(1)$  words of working memory, but it is not clear whether the result can be transferred to  $k_{\text{faults}}$  without increasing the memory usage.

---

<sup>3</sup> A technical report of Long [21] claims that the actual tight bound of the algorithm of Rivest et al. [30] is  $\mathcal{O}(\log n + k)$ , which is consistent with our results.

Finally, we comment on the measures of disorder we adopt in this paper. We study various well-known measures that are mostly folklore. Detailed overviews of measures and their relations were given by Petersson and Moffat [28] and Estivill-Castro and Wood [13]. For the sake of completeness and to get all involved coefficients the full version of this work, accessible at <http://arxiv.org/abs/1702.05932>, provides proofs of all pairwise relations between our parameters; these are depicted in Figure 1.

## 2 Preliminaries

In this paper we consider the following problem: Given an array  $A$  of length  $n$  and an element  $e$ , find the position of  $e$  in  $A$  or report that  $e \notin A$  with as few *queries* as possible. We use  $A[i]$ ,  $i \in 1, \dots, n$  to denote the  $i$ -th entry of  $A$ . We allow access to the entries of  $A$  only via queries to its indices, regarding the relation of the corresponding element to  $e$ . We write  $\text{query}(i)$  for the operation of querying  $A$  at index  $i$ , and let  $\text{query}(i) = '<'$  (respectively,  $'>'$  or  $'='$ ) denote the outcome indicating that  $A[i] < e$  (respectively  $A[i] > e$  or  $A[i] = e$ ). Note that in faulty settings the query outcome need not be accurate.

To keep notation simple, we generally assume the entries of  $A$  to be unique unless explicitly stated otherwise. We emphasize that none of our results relies on this assumption. We can then define  $\text{pos}(a)$  to denote the index of  $a$  in  $A$ , by setting  $\text{pos}(a) = i$  if and only if  $A[i] = a$ . Further, let  $\text{rank}(a) = |\{i : A[i] < a\}| + 1$  be the “correct” position of  $a$  with respect to a sorted copy of  $A$ , irrespective of whether or not  $a \in A$ . We often use an element  $a \in A$  and its index  $\text{pos}(a)$  interchangeably, especially for the target element  $e$ . Note that, as discussed in the introduction, for oblivious algorithms we generally assume  $e \in A$ .

## 3 Searching with imprecise queries

In this section, we consider the problem of finding the index  $\text{pos}(e)$  of an element  $e$  in a sorted array  $A$  of length  $n = 2^d$ ,  $d \in \mathbb{N}$  in a setting where queries may yield erroneous results. We say that  $'<'$  is a lie (the truth) for index  $i$  if  $A[i] \geq e$  ( $A[i] < e$ ), and analogously for  $'>'$  and  $'='$ . To quantify the number of lies, we introduce two parameters  $k_{\text{lies}}$  and  $k_{\text{faults}}$ . The first parameter  $k_{\text{lies}}$  simply bounds the number of queries with erroneous results, which we interpret as the number of lies allowed to an adversary. The second parameter  $k_{\text{faults}}$  bounds the number of indices  $i$  for which  $\text{query}(i)$  (consistently) returns the wrong result, allowing the conclusion that  $e \notin A$  in case  $\text{query}(e)$  yields the wrong result. Equivalently, for an unsorted array  $A$ , we can require all queries to be truthful and define  $k_{\text{faults}}(e)$  to be the number of inversions involving  $e$ , i.e.,  $k_{\text{faults}}(e) = |\{i : (i < \text{pos}(e) \wedge A[i] > e) \vee (i > \text{pos}(e) \wedge A[i] < e)\}|$ . Observe that both definitions of  $k_{\text{faults}}$  are equivalent. For clarity, we write  $k_{\text{faults}}$  when considering the adversarial interpretation, and  $k_{\text{faults}}(e)$  when considering it as a measure of disorder of an unsorted array. For both  $k_{\text{lies}}$  and  $k_{\text{faults}}$ , we only allow queries to  $e$  to yield  $'='$ .

The algorithms of this section operate on the binary search tree rooted at index  $r = n/2$  that contains a path for each possible sequence of queries in a binary search of the array, and identify nodes of the tree with their corresponding indices. We write  $\text{next}_>(i)$  and  $\text{next}_<(i)$  to denote the two successors of node  $i$ , e.g.,  $\text{next}_>(r) = n/4$  and  $\text{next}_<(r) = 3n/4$ . Similarly, we write  $\text{prev}(i)$  to denote the predecessor of  $i$  in the binary search tree, and  $\text{prev}_q(i) = v$  for the last vertex  $v$  on the unique  $r$ - $i$ -path such that  $\text{next}_q(v)$  also lies on the  $r$ - $i$ -path ( $\text{prev}_q(i) = \emptyset$  if no such node exists). Intuitively,  $\text{prev}_q(i)$  is the last vertex corresponding to an array entry larger (if  $q = ">"$ ) or smaller (if  $q = "<"$ ) than  $A[i]$ . For convenience,  $\text{query}(\emptyset) = \emptyset$ ,  $\text{prev}_{</>}(r) = \emptyset$ , and  $\text{next}_{</>}(i) = i$  if  $i$  is a leaf of the tree. We further denote

---

**Algorithm 1:** Algorithm with  $2 \log n + 4k_{\text{lies}}$  queries.

---

```

the algorithm stops once a query yields '='
i ← n/2 // start at the root
while (q ← query(i)) ≠ '=' do // by definition, '=' cannot be a lie
  i' ← prev¬q(i) // ∅ if all queries on the path from the root yielded q
  while i ≠ i' ∧ query(i') = q do // while query(i') contradicts its previous
    outcome...
    | i ← prev(i) // ...backtrack towards i'
  if i ≠ i' then // if we did not backtrack all the way to i'...
    | i ← nextq(i) // ...proceed according to q

```

---

by  $d(i, j)$  the length of the path from node  $i$  to node  $j$  in the search tree. We say that an algorithm *operates on the binary search tree* if no index is queried before its predecessor in the tree.

We start by considering the parameter  $k_{\text{lies}}$ . If we knew the value of this parameter, we could try a regular binary search, replace every query with  $2k_{\text{lies}} + 1$  queries to the same element and use the majority outcome in each step. However, this would give  $(2k_{\text{lies}} + 1) \log n$  queries, where ideally we should not use more than  $\log n + f(k)$  queries. We first give an algorithm that achieves the separation between  $n$  and  $k_{\text{lies}}$  while being oblivious to the value of  $k_{\text{lies}}$ . Importantly, the algorithm only needs  $\mathcal{O}(1)$  memory words, which also makes it applicable to settings where “safe” memory, that cannot be corrupted during the course of the algorithm, is limited. This algorithm still needs  $2 \log n + f(k)$  queries, but we will show later how to build on the same ideas to (almost) eliminate the factor of 2.

Intuitively, Algorithm 1 searches the binary search tree defined above, simply proceeding according to the query outcome at each node. In addition, the algorithm invests queries to double check past decisions. We distinguish left and right turns, depending on whether the algorithm proceeds with the left or the right child. In particular, before proceeding, the algorithm queries the last vertex on the path from the root where it decided for a turn in the opposite direction. While an inconsistency to previous queries is detected, i.e., a query to a vertex where it turned right (or left) gives ‘>’ (or ‘<’), the algorithm backtracks one step. In this manner, the algorithm guarantees that it never proceeds along a wrong path without the adversary investing additional lies. Note that if the algorithm only ever turned right (or left), i.e., there was no previous turn in the opposing direction, it does not double check any past decisions until the query outcome changes. This is alright since either the algorithm is on the right path or the adversary needs to invest a lie in each step.

► **Theorem 1.** *We can find  $e$  obliviously using  $2 \log n + 4k_{\text{lies}}$  queries and  $\mathcal{O}(1)$  memory.*

**Proof.** We claim that Algorithm 1 achieves the bound of the theorem. Note that  $\text{prev}_{\neg q}(i)$  only depends on  $i$  and not on the outcome of previous queries, therefore, we can determine it with  $\mathcal{O}(1)$  memory words. We will show that in each iteration of the outer loop of the algorithm, the potential function  $\Phi = 2d(i, e) + 4k$  decreases by at least one for each query, where  $k$  is the number of remaining lies the adversary may make. This proves the claim, since  $\Phi \geq 0$  and initially  $\Phi \leq 2 \log n + 4k_{\text{lies}}$ . We analyze a single iteration of the outer loop.

Observe that if  $z$  is the number of iterations of the inner loop, then the total number of queries is  $z + 2$  if the inner loop terminates because  $\text{query}(i') = \neg q$ , and  $z + 1$  if it terminates because  $i = i'$ . If an iteration of the inner loop is caused by  $\text{query}(i')$  being a



---

**Algorithm 2:** Algorithm with  $(1 + \frac{1}{c}) \log n + (2c + 2)k_{\text{lies}}$  queries.

---

```

the algorithm stops once a query yields '='
i ← n/2 // start at the root
while (q ← query(i)) ≠ '=' do // by definition, '=' cannot be a lie
  i' ← prev¬q(i) // ∅ if all queries on the path from the root yielded q
  while 0 < cΔi' < d(i, i') + 1 do // while we do not have sufficient
    support to proceed...
    | query(i') // ...query i' for support
  if Δi' = 0 then // if we ran out of support at i' altogether...
    | i ← i' // ...backtrack to i'
  else // if we have sufficient support at i'...
    | i ← nextq(i) // ...proceed according to q

```

---

lie, then in this iteration  $\Delta\Phi \leq 2 - 4 = -2$ , and otherwise,  $d(i, e)$  is decreased by one and likewise  $\Delta\Phi = -2 + 0 = -2$ . Overall, the change in potential during the inner loop is always  $\Delta\Phi = -2z$ . If the inner loop terminates because  $i = i'$ , then  $z \geq 1$  and the total change in potential is  $\Delta\Phi \leq -2z \leq -z - 1$ , enough to cover all  $z + 1$  queries.

Now consider the case that the inner loop terminates because  $\text{query}(i') = \neg q$ . If  $\neg q$  is a lie for  $i'$  or  $q$  is a lie for  $i$ , the adversary invested an additional lie, and even if the last update to  $i$  increases  $d(i, e)$ , the total change in potential is bounded by  $\Delta\Phi \leq -2z - 4 + 2 \leq -2z - 2$ , enough to cover all  $z + 2$  queries. On the other hand, if  $\neg q$  is the truth for  $i'$  and  $q$  is the truth for  $i$ , then  $e \in \{i', \dots, i\}$  and  $i$  must lie on the unique  $r$ - $e$ -path in the search tree (and  $i \neq e$ ). The final update to  $i$  thus decreases  $d(i, e)$  by 1 and the total change in potential is  $\Delta\Phi = -2z - 2$ , again enough to cover all  $z + 2$  queries. ◀

We now adapt Algorithm 1 to minimize the impact of potential lies on the dependency on  $\log n$  in the running time. Intuitively, instead of backing up each query  $q \leftarrow \text{query}(i)$  by a query to  $\text{prev}_{\neg q}(i)$ , we back only one in  $c$  queries (cf. Algorithm 2). During the course of the algorithm and its analysis, we let  $n_{q,j}$  denote the number of queries (so far) to node  $j$  that resulted in  $q \in \{<, >\}$  and  $\Delta_j := |n_{<,j} - n_{>,j}|$ .

► **Theorem 2** (\*<sup>4</sup>). *For every  $c \geq 1$ , we can find  $e$  obliviously using  $(1 + \frac{1}{c}) \log n + (2c + 2)k_{\text{lies}}$  queries.*

**Proof Sketch.** We show that Algorithm 2 achieves the bound of the theorem. Instead of backing up every query as in Algorithm 1, Algorithm 2 only invests one supporting query for every  $c$  consecutive queries with the same outcome (for integral  $c$ ). To capture this, our potential function needs to manage additional potential used to account for these irregular backup queries. This gets more involved, as the algorithm will not usually backtrack by increments of  $c$ , and we need to allow for fractional contributions to the potential in each query. We introduce a potential function of the form

$$\Phi = (1 + \frac{1}{c})d(i, e) + (2 + \frac{1}{c})L + \frac{1}{c}T + (2c + 2)k,$$

where  $k$  is the number of lies remaining to the adversary, and  $L$  and  $T$  will not be formally defined in this proof sketch. Similarly to the proof of Theorem 1, assuming  $\Phi \geq 0$  and

---

<sup>4</sup> Due to space restrictions, proofs for results marked with \* are deferred to the full version of this work.

initially  $L, T = 0$ , we need to show that (on average) the potential function decreases by at least 1 for each query the algorithm makes. We outline intuitively how this is achieved.

The first term of the potential function releases a small amount of potential for every step (in the tree) towards the target element and stores the same amount of potential for every step away from the target. The fourth term releases a lot of potential whenever the adversary invests a lie.

To understand the role of the central terms intuitively, consider an iteration of the algorithm, where it just turned left or right (i.e., moved to a left or right child) and reached node  $i$ , and let  $i'$  be the last vertex where it turned in the opposite direction. Now if the next query result of  $q$  at node  $i$  is a lie, the potential function releases a lot of potential that we can use to pay for the query, the possible backup query, and the change in the other terms of the potential. Let us therefore assume that  $q$  is the truth for  $i$ .

We need to distinguish two situations, depending on whether the turn at  $i'$  was correct or not, i.e., whether the (majority of) backup query outcomes were truthful. If the turn at  $i'$  was wrong, with every step that the algorithm proceeds down the tree, the distance to the target increases, but on the other hand, the adversary needs to invest a lie for each additional backup query. This means that we can afford to store potential with every new backup query, but need to invest potential for every step down the tree. Accordingly, the contribution to  $L$  in the potential function is defined to be  $c\Delta_{j'} - d(j, j')$  and there is no contribution to  $T$ . If the turn at  $i'$  was correct, since  $q$  is also the truth for  $i$  and since the turns at  $i$  and  $i'$  are in opposing directions, the algorithm must still be on a path towards the target  $e$ . Hence, we can store potential with every step down the tree, but have to invest potential to pay for the backup queries. Accordingly, the contribution to  $T$  in the potential function is defined to be  $d(j, j') - c(\Delta_{j'} - 1)$  and there is no contribution to  $L$ .

Now, at some point earlier in the execution of the algorithm,  $i'$  had the role of  $i$  and there was a node  $i''$  with the role of  $i'$ . Since the algorithm may backtrack to  $i'$  in the future, the potential function also needs to remember the contributions of the pair  $(i', i'')$  to  $L$  and  $T$ , as well as the contributions of each earlier such *zig-zag* pair along the path to the root. By carefully defining  $L$  and  $T$ , we can balance all costs in such a way that, in each iteration of the outer loop of Algorithm 2, the potential reduction is at least equal to the number of queries made during that iteration. ◀

To provide a strong lower bound, we restrict ourselves to algorithms that operate on the binary search tree. Such algorithms interpret the array as a binary tree (rooted at entry  $n/2$ , with the two children  $n/4, 3n/4$ , etc.), and never query a node (other than the root) before querying its parent.

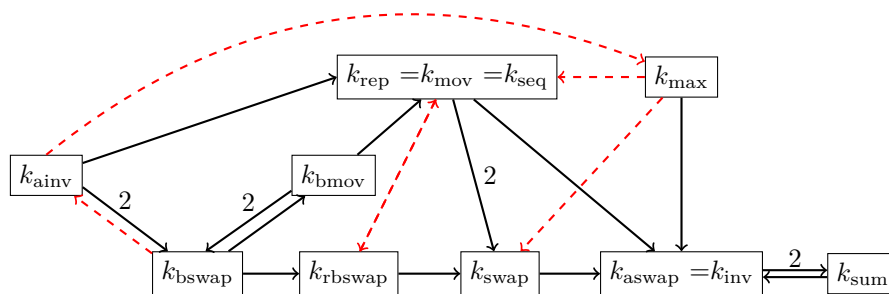
► **Theorem 3 (\*)**. *For every  $c \in \mathbb{N}$  and  $k \in \{k_{\text{lies}}, k_{\text{faults}}\}$ , no algorithm operating on the search tree can find  $e$  with less than  $\log n + ck$  queries in general.<sup>1</sup>*

We show how to translate any algorithm with a performance guarantee with respect to  $k_{\text{lies}}$  to an algorithm with the same guarantee for  $k_{\text{faults}}$ .

► **Theorem 4 (\*)**. *Let  $f : \mathbb{N}^2 \rightarrow \mathbb{N}$ . If we can find  $e$  with  $f(n, k_{\text{lies}})$  queries, then we can find  $e$  with  $f(n, k_{\text{faults}})$  queries.*

## 4 Searching disordered arrays

In this section, we consider the problem of finding the index  $\text{pos}(e)$  of an element  $e$  in array  $A$  of length  $n = 2^d$ ,  $d \in \mathbb{N}$ . In contrast to Section 3, we do not assume  $A$  to be sorted but



■ **Figure 1** Overview of relations between measures of disorder. A solid black path from  $k$  to  $k'$  means that  $k \leq ck'$ , where  $c$  is the product of the edge labels along the path ( $c = 1$  for unlabeled edges). If there is no solid black path from  $k$  to  $k'$ , then  $k$  cannot be bounded by  $ck'$  for any constant  $c$ . Every arc is proved explicitly in the full paper (dashed red arcs correspond to unboundedness results), and *all* other relationships are implied.

expect all queries to yield correct results. We study a variety of parameters that quantify the disorder of  $A$  and provide algorithms and lower bounds with respect to the different parameters. Figure 1 gives the relationship between every pair of parameters.

### 4.1 Bounded displacement

We first consider the two parameters  $k_{\text{sum}}$  and  $k_{\text{max}}$  that quantify the displacement of elements between  $A$  and the sorted counterpart  $A^*$  of  $A$ . More precisely, we define  $k_{\text{sum}} := \sum_{x \in A} |\text{pos}(x) - \text{rank}(x)|$  and  $k_{\text{max}} := \max_{x \in A} |\text{pos}(x) - \text{rank}(x)|$ . We obtain the following bounds.

- ▶ **Theorem 5** (\*).  $\log n/k_{\text{sum}} + 2k_{\text{sum}} + \mathcal{O}(1)$  queries are necessary<sup>1</sup> and sufficient to find  $e$ .
- ▶ **Theorem 6** (\*).  $\log n/k_{\text{max}} + 3k_{\text{max}} + \mathcal{O}(1)$  queries are necessary<sup>1</sup> and sufficient to find  $e$ .

### 4.2 Inversions

We now consider the number of inversions between elements of the array  $A$ . More precisely, we define the number of inversions to be  $k_{\text{inv}} := |\{i < j : A[i] > A[j]\}|$ , and the number of adjacent inversions to be  $k_{\text{ainv}} := |\{i : A[i] > A[i + 1]\}|$ . We have  $k_{\text{sum}} \leq k_{\text{inv}} \leq 2k_{\text{sum}}$ , therefore the results for  $k_{\text{sum}}$  (Theorem 5) carry over to  $k_{\text{inv}}$  with a gap of 2.

- ▶ **Corollary 7**. Every search algorithm needs at least  $\log n/k_{\text{inv}} + 2k_{\text{inv}} + \mathcal{O}(1)$  queries<sup>1</sup>, and we can find  $e$  obliviously with  $\log n/k_{\text{inv}} + 4k_{\text{inv}} + \mathcal{O}(1)$  queries.

In general, we cannot hope to obtain results of similar quality for the smaller parameter  $k_{\text{ainv}}$ ; already for  $k_{\text{ainv}} = 1$  any search algorithm needs to query all  $n$  elements.

- ▶ **Proposition 8**. For  $k_{\text{ainv}} \geq 1$ , no algorithm can find  $e$  with less than  $n$  queries.

Fortunately, we can do much better if the target  $e$  is guaranteed to be in the correct position relative to sorted order, i.e., if  $\text{pos}(e) = \text{rank}(e)$ . Note that this restriction still allows us to prove a lower bound on the necessary number of queries that is much larger than all preceding results. We complement this lower bound by a search algorithm that matches it tightly (up to lower-order terms). Both upper and lower bound hinge on the question of how efficiently (in terms of queries) an algorithm can find a good estimate of  $\text{rank}(e)$  by querying the array.

► **Theorem 9** (\*). *We can find  $e$  using  $2\sqrt{2nk_{\text{ainv}}} + o(\sqrt{nk_{\text{ainv}}})$  queries if  $\text{pos}(e) = \text{rank}(e)$ .*

**Proof Sketch.** We know that if  $e$  is in the array then  $\text{pos}(e) = \text{rank}(e)$ . Since we know neither value beforehand, the algorithm proceeds by determining an estimate of  $\text{rank}(e)$  over two stages of queries. For the first stage, we define a block size  $p = c \cdot \sqrt{n/k_{\text{ainv}}}$  and query every  $(p + 1)$ -st position. This partitions the array into (so far) unqueried blocks of size  $p$ , which we classify into  $\llcorner$ -,  $\lrcorner$ -,  $\langle\rangle$ -, and  $\triangleright$ -blocks according to the query outcomes for the two positions adjacent to the block. Taking into account the number  $k_{\text{ainv}}$  of adjacent inversions, the number of blocks of each type gives rise to an upper and a lower bound on the number of elements that are smaller than  $e$ , and hence on the rank of  $e$ . Essentially, in  $\llcorner$ - and  $\lrcorner$ -blocks with no adjacent inversion there are  $p$  respectively 0 smaller elements, whereas any number between 0 and  $p$  is possible if there is at least one adjacent inversion. In  $\triangleright$ -blocks any number between 0 and  $p$  of smaller elements is possible, but these blocks must contain at least one adjacent inversion. In  $\langle\rangle$ -blocks any number between 0 and  $p$  of smaller elements is possible without adjacent inversions, but the number of  $\langle\rangle$ -blocks can be bounded by the number of  $\triangleright$ -blocks and hence depending on  $k_{\text{ainv}}$ . Overall, this leads to a range of positions that certainly contains the position of  $e$ , if  $e \in A$ . Unfortunately, querying this range entirely would not lead to the claimed upper bound.

Upon second inspection, the unknown number of adjacent inversions in  $\triangleright$ -blocks has the biggest impact on the size of the range. Accordingly, the second stage performs a binary search on each  $\triangleright$ -block, which creates a number of (smaller)  $\lrcorner$ - and  $\llcorner$ -blocks separated by an empty  $\triangleright$ -block between two queried positions (where the binary search terminates). Thus, repeating the above analysis after the refinement, all  $\triangleright$ -blocks are now of size zero and their impact on the range of possible positions is reduced. The algorithm can now afford to query this range entirely and either find  $e$  or be sure that it is not in  $A$ . The claimed bound is obtained by choosing the value of  $c$  in the block size  $p$  in order to balance the cost of making the initial queries and the cost of eventually querying the computed range. ◀

► **Theorem 10** (\*). *Every search algorithm needs at least  $2\sqrt{2nk_{\text{ainv}}} - o(\sqrt{nk_{\text{ainv}}})$  queries<sup>1</sup>, even if  $\text{pos}(e) = \text{rank}(e)$ .*

**Proof Sketch.** We outline an adversarial strategy for replying to queries. This strategy needs to avoid revealing the position of  $e$  before the claimed number of queries, and simultaneously needs to maintain that at least one realization of the array remains that is consistent with the replies made so far, and that has at most  $k_{\text{ainv}}$  adjacent inversions and  $\text{pos}(e) = \text{rank}(e)$ .

At the beginning, our strategy replies with ‘ $\langle$ ’ to queries in the first half of  $A$  and with ‘ $\rangle$ ’ to queries in the second half. At some point, we may not be able to place  $e$  in one of the two halves anymore: Say that the rightmost unqueried position in the left half has  $p$  positions on its right for which we already committed to ‘ $\langle$ ’. Since  $\text{pos}(e) = \text{rank}(e)$ , the number of elements smaller than  $e$  and on its right must be equal to the number of larger elements on its left. Hence, a realization with  $e$  in the left half can only exist if there still remains a way of placing  $p$  elements larger than  $e$  in the left half (leaving the rightmost unqueried position for  $e$ ), without causing more than  $k_{\text{ainv}}$  adjacent inversions. This is the case exactly if we can still find a set of (roughly) at most  $k_{\text{ainv}}$  disjoint blocks of consecutive, unqueried positions in the left half, which contain at least  $p$  unqueried positions overall. Analogously, at some point, we may not be able to place  $e$  in the right half anymore. Our strategy continues to reply to queries as before, until it has to commit to a realization and reply accordingly. Careful analysis of the strategy yields that committing to a position for  $e$  can be avoided until the claimed number of queries have been invested. ◀

### 4.3 Edit distances

We now consider parameters that bound the number of elementary array modifications needed to sort the given array  $A$ . More precisely, a *replacement* is the operation of replacing one element with a new element, and we let  $k_{\text{rep}}$  be the (minimum) number of replacements needed to obtain a sorted array. A *swap* is the exchange of the content of two array positions, and  $k_{\text{swap}}$  is the number of swaps needed to sort  $A$ . We let  $k_{\text{aswap}}$  be the number of swaps of pairs of neighboring elements needed to sort  $A$ . A *move* is the operation of removing an element and re-inserting it after a given position  $i$ , shifting all elements between old and new position by one. We let  $k_{\text{mov}}$  be the number of moves needed to sort  $A$ .

Clearly, starting from a sorted array, we can move  $e$  to any position, without using more than a single move or swap, or two replacements involving  $e$ . To find  $e$  we then have to query the entire array.

► **Proposition 11.** *For  $k_{\text{mov}} \geq 1$ ,  $k_{\text{swap}} \geq 1$ , or  $k_{\text{rep}} \geq 2$ , no algorithm can find  $e$  with less than  $n$  queries in general.*

We can obtain significantly improved bounds if the element  $e$  remains at its correct position relative to the sorted array. Recall that we can interpret  $k_{\text{faults}}$  as a measure of disorder via  $k_{\text{faults}}(e) = |i : (i < \text{pos}(e) \wedge A[i] > e) \vee (i > \text{pos}(e) \wedge A[i] < e)|$ .

► **Lemma 12 (\*)**. *If  $\text{rank}(e) = \text{pos}(e)$ , then  $k_{\text{faults}}(e) \leq \min\{2k_{\text{rep}}, 4k_{\text{swap}}, 2k_{\text{mov}}\}$ .*

With this lemma, we can translate the upper bounds of any algorithm for  $k_{\text{lies}}$ . Before we do, we introduce another measure of disorder, that turns out to be closely related to  $k_{\text{mov}}$ . We define the parameter  $k_{\text{seq}}$  to be such that  $n - k_{\text{seq}}$  is the length of a longest nondecreasing subsequence in  $A$ . It turns out that  $k_{\text{mov}} = k_{\text{seq}}$ , and we can thus include this parameter in our upper bound.

► **Theorem 13 (\*)**. *Let  $f: \mathbb{N}^2 \rightarrow \mathbb{N}$ . If  $\text{rank}(e) = \text{pos}(e)$  and we can find  $e$  with  $f(n, k_{\text{lies}})$  queries, then we can find  $e$  with  $\min\{f(n, 2k_{\text{rep}}), f(n, 4k_{\text{swap}}), f(n, 2k_{\text{mov}}), f(n, 2k_{\text{seq}})\}$  queries.*

We can also carry over the lower bound from parameter  $k_{\text{faults}}$ .

► **Corollary 14.** *For every  $c \in \mathbb{N}$  and  $\text{pos}(e) = \text{rank}(e)$ , no algorithm operating on the search tree can find  $e$  with less than  $\log n + ck$  queries in general, for  $k \in \{k_{\text{rep}}, k_{\text{swap}}, k_{\text{mov}}, k_{\text{seq}}\}$ .<sup>1</sup>*

Finally, we immediately obtain bounds for  $k_{\text{aswap}}$  from Corollary 7, because  $k_{\text{aswap}} = k_{\text{inv}}$ .

► **Corollary 15.** *Every search algorithm needs at least  $\log n/k_{\text{aswap}} + 2k_{\text{aswap}} + \mathcal{O}(1)$  queries<sup>1</sup>, and we can find  $e$  obviously with  $\log n/k_{\text{aswap}} + 4k_{\text{aswap}} + \mathcal{O}(1)$  queries.*

### 4.4 Block edit distances

In this section we consider the parameters  $k_{\text{bswap}}$ ,  $k_{\text{rbswap}}$ , and  $k_{\text{bmov}}$ , which bound the number of block edit operations needed to sort  $A$ . A *block* is defined to be a subarray  $A[i, i+1, \dots, j]$  of consecutive elements. A *block swap* is the operation of exchanging a subarray  $A[i, \dots, j]$  with a subarray  $A[i', \dots, j']$  and vice versa, where  $i < j < i' < j'$ . Note that a block swap may affect the positions of other elements in case that the two blocks are of different sizes. The parameter  $k_{\text{bswap}}$  bounds the number of block swaps needed to sort  $A$ . For  $k_{\text{rbswap}}$  we only allow block swaps restricted to pairs of blocks of equal sizes. Finally, for  $k_{\text{bmov}}$  one of the two blocks must be empty, i.e., only block moves are allowed. For all three parameters one can easily prove that, without further restrictions, search algorithms need to query all positions of an array to find the target.

► **Proposition 16.** *For  $k_{\text{bswap}} \geq 1$ ,  $k_{\text{rbswap}} \geq 1$ , or  $k_{\text{bmov}} \geq 1$ , no algorithm can find  $e$  with less than  $n$  queries.*

Complementing this lower bound, for all of three parameters, an upper bound of  $\mathcal{O}(\sqrt{nk})$  for finding  $e$  when  $\text{pos}(e) = \text{rank}(e)$  follows immediately from the results for  $k_{\text{ainv}}$  of Section 4.2 and the fact that  $k_{\text{ainv}} \leq 2k_{\text{bswap}}$  and  $k_{\text{bswap}} \leq \min\{k_{\text{rbswap}}, k_{\text{bmov}}\}$ . By inspecting the upper and lower bounds proved for  $k_{\text{ainv}}$ , and adapting the proofs, we are able to obtain tight leading constants in the upper and lower bounds for  $k_{\text{bmov}}$  and  $k_{\text{bswap}}$ , and leading constants within a factor of  $\sqrt{2}$  for  $k_{\text{rbswap}}$ .

► **Theorem 17** (\*). *If  $\text{pos}(e) = \text{rank}(e)$ , then  $4\sqrt{nk_{\text{bswap}}} + o(\sqrt{nk_{\text{bswap}}})$  queries are necessary<sup>1</sup> and sufficient to find  $e$ .*

► **Theorem 18** (\*). *If  $\text{pos}(e) = \text{rank}(e)$ , then  $2\sqrt{2nk_{\text{rbswap}}} + o(\sqrt{nk_{\text{rbswap}}})$  queries are necessary<sup>1</sup> and  $4\sqrt{nk_{\text{rbswap}}} + o(\sqrt{nk_{\text{rbswap}}})$  queries are sufficient to find  $e$ .*

► **Theorem 19** (\*). *If  $\text{pos}(e) = \text{rank}(e)$ , then  $2\sqrt{2nk_{\text{bmov}}} + o(\sqrt{nk_{\text{bmov}}})$  queries are necessary<sup>1</sup> and sufficient to find  $e$ .*

## 5 Conclusion

We presented upper and lower bounds for the worst-case query complexity of comparison-based search algorithms that are robust to persistent and temporary read errors, or are adaptive to partially disordered input arrays. For many cases we gave algorithms that are optimal up to lower order terms. In addition, many of the algorithms are oblivious to the value of the parameter quantifying errors/disorder, assuming the target element is present in the array. In most cases, for small values of  $k$ , the dependence of our algorithms on the number  $n$  of elements is close to  $\log n$ , with only additive dependency on the number of imprecisions. In other words, these results smoothly interpolate between parameter regimes where algorithms are as good as binary search and the unavoidable worst-case where linear search is best possible.

That said, why should one be interested in, e.g., almost tight bounds relative to the number of block moves that take  $A$  to a sorted array, as the bounds are far from binary search? The point is that only the total number of comparisons matter, and having a worse function that depends on a (in this case) much smaller parameter value can be favorable to having a much better function of a large parameter value. E.g., after a constant number of block swaps the parameters  $k_{\text{max}}$ ,  $k_{\text{sum}}$  etc. may have value  $\Omega(n)$  and the guaranteed bound becomes trivial, while running the search algorithm for the case of few block swaps guarantees  $\mathcal{O}(\sqrt{n})$  comparisons. Similarly, having tight bounds for the various parameters gives us the exact (worst-case) regime for the chosen parameter (in terms of  $n$ ) where a sophisticated algorithm can outperform linear search, or even be as good as binary search.

Despite having already asymptotic tightness, it would be interesting to close the gaps between coefficients of dominant terms in upper and lower bounds for some of the cases. Another question would be to find a different restriction than  $\text{pos}(e) = \text{rank}(e)$ , i.e., the target being in the correct position relative to sorted order, that avoids degenerate lower bounds of  $\Omega(n)$  queries for several parameters. A relaxation to allowing a target displacement of  $\ell$  and giving cost in terms of  $n$ ,  $k$ , and  $\ell$  seems doable in most cases, but is unlikely to be particularly insightful. Finally, it seems interesting to study whether randomization could lead to improved algorithms for some of the cases. The analysis of randomized lower bounds requires entirely new adversarial strategies since the adversary must choose an instantiation without access to the random bits of the algorithm.

**Acknowledgements.** The authors are grateful to several reviewers for their helpful remarks regarding presentation and pertinent literature references.

---

## References

- 1 Arne Andersson, Torben Hagerup, Johan Håstad, and Ola Petersson. Tight bounds for searching a sorted array of strings. *SIAM J. Comput.*, 30(5):1552–1578, 2000. doi:10.1137/S0097539797329889.
- 2 Stanislav Angelov, Keshav Kunal, and Andrew McGregor. Sorting and selection with random costs. In *Proceedings of the 8th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 48–59, 2008. doi:10.1007/978-3-540-78773-0\_5.
- 3 Javed A. Aslam and Aditi Dhagat. Searching in the presence of linearly bounded errors. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 486–493, 1991.
- 4 Jérémy Barbay and Gonzalo Navarro. On compressing permutations and adaptive sorting. *Theor. Comput. Sci.*, 513:109–123, 2013. doi:10.1016/j.tcs.2013.10.019.
- 5 Therese C. Biedl, Timothy M. Chan, Erik D. Demaine, Rudolf Fleischer, Mordecai J. Golin, James A. King, and J. Ian Munro. Fun-sort—or the chaos of unordered binary search. *Discrete Applied Mathematics*, 144(3):231–236, 2004. doi:10.1016/j.dam.2004.01.003.
- 6 Biagio Bonasera, Emilio Ferrara, Giacomo Fiumara, Francesco Pagano, and Alessandro Proveti. Adaptive search over sorted sets. *J. Discrete Algorithms*, 30:128–133, 2015. doi:10.1016/j.jda.2014.12.007.
- 7 Ryan S. Borgstrom and S. Rao Kosaraju. Comparison-based search in the presence of errors. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 130–136, 1993. doi:10.1145/167088.167129.
- 8 Allan Borodin, Leonidas J. Guibas, Nancy A. Lynch, and Andrew Chi-Chih Yao. Efficient searching using partial ordering. *Inf. Process. Lett.*, 12(2):71–75, 1981. doi:10.1016/0020-0190(81)90005-3.
- 9 Gerth Stølting Brodal, Rolf Fagerberg, Irene Finocchi, Fabrizio Grandoni, Giuseppe F. Italiano, Allan Grønlund Jørgensen, Gabriel Moruz, and Thomas Mølhave. Optimal resilient dynamic dictionaries. In *Proceedings of the 15th Annual European Symposium on Algorithms (ESA)*, pages 347–358, 2007.
- 10 F. Warren Burton and Gilbert N. Lewis. A robust variation of interpolation search. *Inf. Process. Lett.*, 10(4/5):198–201, 1980. doi:10.1016/0020-0190(80)90139-8.
- 11 Ferdinando Cicalese. *Fault-Tolerant Search Algorithms – Reliable Computation with Unreliable Information*. Springer, 2013. doi:10.1007/978-3-642-17327-1.
- 12 Aditi Dhagat, Peter Gacs, and Peter Winkler. On playing “twenty questions” with a liar. In *Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 16–22, 1992.
- 13 Vladimir Estivill-Castro and Derick Wood. A survey of adaptive sorting algorithms. *ACM Comput. Surv.*, 24(4):441–476, 1992. doi:10.1145/146370.146381.
- 14 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994.
- 15 Irene Finocchi, Fabrizio Grandoni, and Giuseppe F. Italiano. Optimal resilient sorting and searching in the presence of memory faults. *Theor. Comput. Sci.*, 410(44):4457–4470, 2009. doi:10.1016/j.tcs.2009.07.026.
- 16 Irene Finocchi and Giuseppe F. Italiano. Sorting and searching in faulty memories. *Algorithmica*, 52(3):309–332, 2008. doi:10.1007/s00453-007-9088-4.
- 17 Gianni Franceschini and Roberto Grossi. No sorting? better searching! *ACM Transactions on Algorithms*, 4(1), 2008. doi:10.1145/1328911.1328913.

- 18 Michael L. Fredman. The number of tests required to search an unordered table. *Inf. Process. Lett.*, 87(2):85–88, 2003. doi:10.1016/S0020-0190(03)00260-6.
- 19 Anupam Gupta and Amit Kumar. Sorting and selection with structured costs. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, (FOCS)*, pages 416–425, 2001. doi:10.1109/SFCS.2001.959916.
- 20 Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.
- 21 Philip M. Long. Sorting and searching with a faulty comparison oracle. Technical Report UCSC-CRL-92-15, University of California at Santa Cruz, 1992.
- 22 Harry G. Mairson. Average case lower bounds on the construction and searching of partial orders. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 303–311, 1985. doi:10.1109/SFCS.1985.13.
- 23 Kurt Mehlhorn. Sorting presorted files. In *Proceedings of the 4th GI-Conference on Theoretical Computer Science*, pages 199–212, 1979. doi:10.1007/3-540-09118-1\_22.
- 24 Kurt Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*, volume 1 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1984. URL: <http://www.mpi-sb.mpg.de/~mehlhorn/DataAlgbbooks.html>.
- 25 S. Muthukrishnan. On optimal strategies for searching in the presence of errors. In *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 680–689, 1994.
- 26 Andrzej Pelc. Searching with known error probability. *Theoretical Computer Science*, 63(2):185–202, 1989.
- 27 Andrzej Pelc. Searching games with errors – fifty years of coping with liars. *Theoretical Computer Science*, 270(1-2):71–109, 2002.
- 28 Ola Petersson and Alistair Moffat. A framework for adaptive sorting. *Discrete Applied Mathematics*, 59(2):153–179, 1995. doi:10.1016/0166-218X(93)E0160-Z.
- 29 Erez Petrank and Guy N. Rothblum. Selection from structured data sets. *Electronic Colloquium on Computational Complexity (ECCC)*, 2004. TR04-085. URL: <http://eccc.hpi-web.de/eccc-reports/2004/TR04-085/index.html>.
- 30 Ronald L. Rivest, Albert R. Meyer, Daniel J. Kleitman, Karl Winkmann, and Joel Spencer. Coping with errors in binary search procedures. *Journal of Computer and System Sciences*, 20(3):396–404, 1980.
- 31 Robert Sedgewick. *Algorithms in C++ – Parts 1–4: Fundamentals, Data Structures, Sorting, Searching*. Addison-Wesley-Longman, 1998.
- 32 Andrew Chi-Chih Yao. Should tables be sorted? *J. ACM*, 28(3):615–628, 1981. doi:10.1145/322261.322274.



# Graphic TSP in Cubic Graphs\*

Zdeněk Dvořák<sup>1</sup>, Daniel Král<sup>2</sup>, and Bojan Mohar<sup>3</sup>

- 1 Computer Science Institute, Charles University, Prague, Czech Republic  
rakdver@iuuk.mff.cuni.cz
- 2 Mathematics Institute, DIMAP and Department of Computer Science,  
University of Warwick, Coventry, UK  
d.kral@warwick.ac.uk
- 3 Department of Mathematics, Simon Fraser University, Burnaby, Canada  
mohar@sfu.ca

---

## Abstract

We present a polynomial-time  $9/7$ -approximation algorithm for the graphic TSP for cubic graphs, which improves the previously best approximation factor of 1.3 for 2-connected cubic graphs and drops the requirement of 2-connectivity at the same time. To design our algorithm, we prove that every simple 2-connected cubic  $n$ -vertex graph contains a spanning closed walk of length at most  $9n/7 - 1$ , and that such a walk can be found in polynomial time.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** Graphic TSP, approximation algorithms, cubic graphs

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.27

## 1 Introduction

The Travelling Salesperson Problem (TSP) is one of the most central problems in combinatorial optimization. The problem asks to find a shortest closed walk visiting each vertex at least once in an edge-weighted graph, or alternatively to find a shortest Hamilton cycle in a complete graph where the edge weights satisfy the triangle inequality. The Travelling Salesperson Problem is notoriously hard. The approximation factor of  $3/2$  established by Christofides [4] has not been improved for 40 years despite a significant effort of many researchers. The particular case of the problem, the Hamilton Cycle Problem, was among the first problems to be shown to be NP-hard. Moreover, Karpinski, Lampis and Schmied [10] have recently shown that the Travelling Salesperson Problem is NP-hard to approximate within the factor  $123/122$ , improving the earlier inapproximability results of Lampis [12] and of Papadimitriou and Vempala [18]. In this paper, we are concerned with an important special case of the Travelling Salesperson Problem, the graphic TSP, which asks to find a shortest closed walk visiting each vertex at least once in a graph where all edges have unit weight. We will refer to such a walk as to a *TSP walk*.

There has recently been a lot of research focused on approximation algorithms for the graphic TSP, which was ignited by the breakthrough of the  $3/2$ -approximation barrier in

---

\* The first author was supported by the Center of Excellence – Institute for Theoretical Computer Science, project P202/12/G061 of Czech Science Foundation. The work of the second author has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 648509). This publication reflects only its authors’ view; the European Research Council Executive Agency is not responsible for any use that may be made of the information it contains. The third author was supported in part by an NSERC Discovery Grant (Canada), by the Canada Research Chairs program, and by a Research Grant of ARRS (Slovenia).



the case of 3-connected cubic graphs by Gamarnik, Lewenstein and Sviridenko [7]. This was followed by the improvement of the  $3/2$ -approximation factor for the general graphic TSP by Oveis Gharan, Saberi and Singh [16]. Next, Mömke and Svensson [14] designed a  $1.461$ -approximation algorithm for the problem and Mucha [15] showed that their algorithm is actually a  $13/9$ -approximation algorithm. This line of research culminated with the  $7/5$ -approximation algorithm of Sebö and Vygen [19].

We here focus on the case of graphic TSP for cubic graphs, which was at the beginning of this line of improvements. The  $(3/2 - 5/389)$ -approximation algorithm of Gamarnik et al. [7] for 3-connected cubic graphs was improved by Aggarwal, Garg and Gupta [1], who designed a  $4/3$ -approximation algorithm. Next, Boyd et al. [2] found a  $4/3$ -approximation algorithm for 2-connected cubic graphs. The barrier of the  $4/3$ -approximation factor was broken by Correa, Larré and Soto [5] who designed a  $(4/3 - 1/61236)$ -approximation algorithm for this class of graphs. The currently best algorithm for 2-connected cubic graphs is the  $1.3$ -approximation algorithm of Candráková and Lukot'ka [3], based on their result on the existence of a TSP walk of length at most  $1.3n - 2$  in 2-connected cubic  $n$ -vertex graphs. We improve this result as follows. Note that we obtain a better approximation factor and Theorem 2 also applies to a larger class of graphs.

► **Theorem 1.** *There exists a polynomial-time algorithm that for a given 2-connected subcubic  $n$ -vertex graph with  $n_2$  vertices of degree two outputs a TSP walk of length at most*

$$\frac{9}{7}n + \frac{2}{7}n_2 - 1.$$

► **Theorem 2.** *There exists a polynomial-time  $9/7$ -approximation algorithm for the graphic TSP for cubic graphs.*

Note that our approximation factor matches the approximation factor for cubic bipartite graphs in the algorithm Karp and Ravi [9], who designed a  $9/7$ -approximation algorithm for the graphic TSP for cubic bipartite graphs. However, van Zuylen [20] has recently found a  $5/4$ -approximation algorithm for this class of graphs. Both the result of Karp and Ravi, and the result of van Zuylen are based on finding a TSP walk of length of at most  $9n/7$  and  $5n/4$ , respectively, in an  $n$ -vertex cubic bipartite graph. On the negative side, Karpinski and Schmied [11] showed that the graphic TSP is NP-hard to approximate within the factor of  $535/534$  in the general case and within the factor  $1153/1152$  in the case of cubic graphs.

Our contribution in addition to improving the approximation factor for graphic TSP for cubic graphs is also in bringing several new ideas to the table. The proof of our main result, Theorem 1, differs from the usual line of proofs in this area. In particular, to establish the existence of a TSP walk of length at most  $9n/7 - 1$  in a 2-connected cubic  $n$ -vertex graph, we allow subcubic graphs as inputs and perform reductions in this larger class of graphs. While we cannot establish the approximation factor of  $9/7$  for this larger class of graphs, we are still able to show that our technique yields the existence of a TSP walk of length at most  $9n/7 - 1$  for cubic  $n$ -vertex graphs. At this point, we should remark that we have not attempted to optimize the running time of our algorithm.

We conclude with a brief discussion on possible improvements of the bound from Theorem 1. In Section 5, we give a construction of a 2-connected cubic  $n$ -vertex graph with no TSP walks of length smaller than  $\frac{5}{4}n - 2$  (Proposition 16) and a 2-connected subcubic  $n$ -vertex graph with  $n_2 = \Theta(n)$  vertices of degree two with no TSP walks of length smaller than  $\frac{5}{4}n + \frac{1}{4}n_2 - 1$  (Proposition 14); the former construction was also found independently by Mazák and Lukot'ka [13]. We believe that these two constructions provide the tight examples

for an improvement of Theorem 1 and conjecture the following. We also refer to a more detailed discussion at the end of Section 5.

► **Conjecture 3.** *Every 2-connected subcubic  $n$ -vertex graph with  $n_2$  vertices of degree two has a TSP walk of length at most*

$$\frac{5}{4}n + \frac{1}{4}n_2 - 1.$$

Note that Conjecture 3 is of interest for small values of  $n_2$ ; in particular, its statement is known to be true if  $n_2 \geq n/3$  [14]. We would like to stress that it is important that Conjecture 3 deals with simple graphs, i.e., graphs without parallel edges. Indeed, consider the cubic graph  $G$  obtained as follows: start with the graph that has two vertices of degree three that are joined by three paths, each having  $2\ell$  internal vertices of degree two, and replace every second edge of these paths with a pair of parallel edges to get a cubic graph. The graph  $G$  has  $n = 6\ell + 2$  vertices but no TSP walk of length shorter than  $8\ell + 2$ .

## 2 Preliminaries

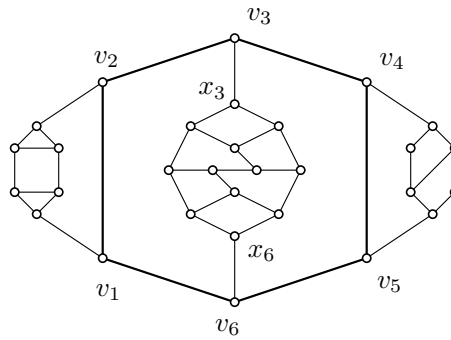
In this section, we fix the notation used in the paper and make several simple observations on the concepts that we use.

All graphs considered in this paper are *simple*, i.e., they do not contain parallel edges. When we allow parallel edges, we will always emphasize this by referring to a considered graph as to a *multigraph*. We will occasionally want to stress that a graph obtained during the proof has no parallel edges and we will do so by saying that it is simple even if saying so is superfluous. The *underlying graph* of a multigraph  $H$  is the graph obtained from  $H$  by suppressing parallel edges, i.e., replacing each set of parallel edges by a single edge.

If  $G$  is a graph, its vertex set is denoted by  $V(G)$  and its edge set by  $E(G)$ . Further, the number of vertices of  $G$  is denoted by  $n(G)$  and the number of its vertices of degree two by  $n_2(G)$ . If  $w$  a vertex of  $G$ , then  $G - w$  is a graph obtained by deleting the vertex  $w$  and all the edges incident with  $w$ . Similarly, if  $W$  is a set of vertices of  $G$ , then  $G - W$  is the graph obtained by deleting all vertices of  $W$  and edges incident with them. Finally, if  $F$  is a set of its edges, then  $G \setminus F$  is the graph obtained from  $G$  by removing the edges of  $F$  but none of the vertices.

A graph with all vertices of degree at most three is called *subcubic*. We say that a graph  $G$  is  *$k$ -connected* if it has at least  $k + 1$  vertices and  $G - W$  is connected for any  $W \subseteq V(G)$  containing at most  $k - 1$  vertices (here, we deviate from the standard terminology since we do not consider  $K_2$  to be 2-connected). If  $G$  is connected but not 2-connected, then a vertex  $v$  such that  $G - v$  is not connected is called a *cut-vertex*. Maximal 2-connected subgraphs of  $G$  and edges that are not contained in any 2-connected subgraphs of  $G$  are called *blocks*. A subset  $F$  of the edges of a graph  $G$  is an *edge-cut* if the graph  $G \setminus F$  have more components than  $G$  and  $F$  is minimal with this property. Such a subset  $F$  containing exactly  $k$  edges will also be referred to as  *$k$ -edge-cut*. An edge forming a 1-edge-cut is called a *cut-edge*. A graph  $G$  is  *$k$ -edge-connected* if it has no  $\ell$ -edge-cut for  $\ell < k$ . Note that a subcubic graph  $G$  with at least two vertices is 2-connected if and only if it is 2-edge-connected.

A  *$\theta$ -graph* is a simple graph obtained from the pair of vertices joined by three parallel edges by subdividing some of the edges several times. In other words, a  $\theta$ -graph is a graph that contains two vertices of degree three joined by three paths formed by vertices of degree two such that at most one of these paths is trivial, i.e., it is a single edge. In our consideration, we will need to consider a special type of cycles of length six in subcubic graphs, which



■ **Figure 1** A  $\theta$ -cycle with poles  $v_3$  and  $v_6$ .

resembles  $\theta$ -graphs. A cycle  $K = v_1 \dots v_6$  of length six in a subcubic graph  $G$  is a  $\theta$ -cycle, if all vertices  $v_1, \dots, v_6$  have degree three, their neighbors  $x_1, \dots, x_6$  outside of  $K$  are pairwise distinct, and  $G - V(K)$  has three connected components, one containing  $x_1$  and  $x_2$ , one containing  $x_4$  and  $x_5$ , and one containing  $x_3$  and  $x_6$ . See Figure 1 for an example. The vertices  $v_3$  and  $v_6$  of the cycle  $K$  will be referred to as the *poles* of the  $\theta$ -cycle  $K$ .

We say that a multigraph is *Eulerian* if all its vertices have even degree; note that we do not require the multigraph to be connected, i.e., a multigraph has an Eulerian tour if and only if it is Eulerian and connected. A subgraph is *spanning* if it contains all vertices of the original graphs, possibly some of them as isolated vertices, i.e., vertices of degree zero. It is easy to relate the length of the shortest TSP walk in a graph  $G$  to the size of Eulerian multigraphs using edges of  $G$  as follows. To simplify our presentation, let  $\text{tsp}(G)$  denote the length of the shortest TSP walk in a graph  $G$ . The proof of the next observation is omitted because of the space constraints.

► **Observation 4.** *For every graph  $G$ ,  $\text{tsp}(G)$  is equal to the minimum number of edges of a connected Eulerian multigraph  $H$  such that the underlying graph of  $H$  is a spanning subgraph of  $G$ .*

We now explore the link between Eulerian spanning subgraphs and the minimum length of a TSP walk further. For a graph  $G$ , let  $c(F)$  denote the number of non-trivial components of  $F$ , i.e., components formed by two or more vertices, and let  $i(F)$  be the number of isolated vertices of  $F$ . We define the *excess* of a graph  $F$  as

$$\text{exc}(F) = 2c(F) + i(F).$$

If  $G$  is a subcubic graph, we define

$$\text{minexc}(G) = \min \{ \text{exc}(F) : F \text{ spanning Eulerian subgraph of } G \}.$$

Note that any subcubic Eulerian graph  $F$  is a union of  $c(F)$  cycles and  $i(F)$  isolated vertices, i.e., the spanning subgraph  $F$  of a subcubic graph  $G$  with  $\text{exc}(F) = \text{minexc}(G)$  must also have this structure. The values of  $\text{minexc}(G)$  for simple-structured graphs are given in the next observation.

► **Observation 5.** *The following holds.*

1. *If  $G$  is a cycle, then  $\text{minexc}(G) = 2 < \frac{n(G)+n_2(G)}{4} + 1$ .*
2. *If  $G = K_4$ , then  $\text{minexc}(G) = 2 = \frac{n(G)+n_2(G)}{4} + 1$ .*

3. If  $G$  is  $\theta$ -graph with  $k_1$ ,  $k_2$  and  $k_3$  vertices of degree two on the paths joining its two vertices of degree three and  $k_1 \leq k_2 \leq k_3$  (note that  $k_2 \neq 0$  by the definition of a  $\theta$ -graph),  $\text{minexc}(G) = 2 + k_1 \leq \frac{n(G) + n_2(G)}{4} + 1$ .

We next relate the quantity  $\text{minexc}(G)$  to the length of the shortest TSP walk in  $G$ .

► **Observation 6.** Let  $G$  be a connected subcubic  $n$ -vertex graph, and let  $F$  be a spanning Eulerian subgraph  $F$  of  $G$ . There exists a polynomial-time algorithm that finds a TSP walk of length  $n - 2 + \text{exc}(F)$ . In addition, the minimum length of a TSP walk in  $G$  is equal to

$$\text{tsp}(G) = n - 2 + \text{minexc}(G).$$

**Proof.** Let  $F$  be a spanning Eulerian subgraph of  $G$ . We aim to construct a TSP walk of length  $n - 2 + \text{exc}(F)$ . The subgraph  $F$  has  $c(F) + i(F)$  components. Since  $F$  is subcubic, each of the  $c(F)$  non-trivial components of  $F$  is a cycle, which implies that  $F$  has  $n - i(F)$  edges. Since  $G$  is connected, there exists a subset  $S$  of the edges of  $G$  such that  $|S| = c(F) + i(F) - 1$  and  $F$  together with the edges of  $S$  is connected. Clearly, such a subset  $S$  can be found in linear time. Let  $H$  be the multigraph obtained from  $F$  by adding each edge of  $S$  with multiplicity two. Since  $H$  is a connected Eulerian multigraph whose underlying graph is a spanning subgraph of  $G$ , the proof of Observation 4 yields that it corresponds to an Eulerian tour of length

$$|E(H)| = |E(F)| + 2|S| = n - i(F) + 2(c(F) + i(F) - 1) = n - 2 + \text{exc}(F),$$

which can be found in linear time. In particular, it holds that  $\text{tsp}(G) \leq n - 2 + \text{exc}(F)$ . Since the choice of  $F$  was arbitrary, we conclude that  $\text{tsp}(G) \leq n - 2 + \text{minexc}(G)$ .

To finish the proof, we need to show that  $n - 2 + \text{minexc}(G) \leq \text{tsp}(G)$ . By Observation 4, there exists a connected Eulerian multigraph  $H$  with  $|E(H)| = \text{tsp}(G)$  such that its underlying graph is a spanning subgraph of  $G$ . By the minimality of  $|E(H)|$ , every edge of  $H$  has multiplicity at most two (otherwise, we can decrease its multiplicity by 2 while keeping the multigraph Eulerian and connected). Similarly, removing any pair of parallel edges of  $H$  disconnects  $H$  (as the resulting multigraph would still be Eulerian), i.e., the edge in the underlying graph of  $H$  corresponding to a pair of parallel edges is a cut-edge. Let  $F$  be the graph obtained from  $H$  by removing all the pairs of parallel edges. The number of components of  $F$  is equal to

$$c(F) + i(F) = \frac{|E(H)| - |E(F)|}{2} + 1.$$

Since  $F$  is subcubic, it is a union of  $c(F)$  cycles and  $i(F)$  isolated vertices, which implies that  $|E(F)| = n - i(F)$ . Consequently, we get that

$$c(F) + i(F) = \frac{|E(H)| - (n - i(F))}{2} + 1,$$

which yields the desired inequality

$$n - 2 + \text{minexc}(G) \leq n - 2 + \text{exc}(F) = n - 2 + 2c(F) + i(F) = |E(H)| = \text{tsp}(G). \quad \blacktriangleleft$$

### 3 Reductions

In this section, we present a way of reducing a 2-connected subcubic graph to a smaller one such that a spanning Eulerian subgraph of the smaller graph yields a spanning Eulerian

subgraph of the original graph with few edges. We now define this process more formally. For subcubic graphs  $G$  and  $G'$ , let

$$\delta(G, G') = (n(G) + n_2(G)) - (n(G') + n_2(G')) .$$

We say that a 2-connected subcubic graph  $G'$  is a reduction of a 2-connected subcubic graph  $G$  if  $n(G') < n(G)$ ,  $\delta(G, G') \geq 0$ , and there exists a linear-time algorithm that turns any spanning Eulerian subgraph  $F'$  of  $G'$  into a spanning Eulerian subgraph  $F$  of  $G$  satisfying

$$\text{exc}(F) \leq \text{exc}(F') + \frac{1}{4} \cdot \delta(G, G') . \tag{1}$$

For the proof of our main result, it would be enough to prove the lemmas in this section with  $\frac{1}{4}$  replaced by  $\frac{2}{7}$  in (1). However, this would not simplify most of our arguments and we believe that the stronger form of (1) can be useful in an eventual proof of Conjecture 3.

The reductions that we present are intended to be applied to an input subcubic 2-connected graph until the resulting graph is simple or it has a special structure. A subcubic 2-connected graph is *basic* if it is a cycle, a  $\theta$ -graph, or  $K_4$ . A subcubic 2-connected graph that is not basic will be referred to as *non-basic*. The reductions involve altering a subgraph  $K$  of a graph  $G$  such that  $K$  has some additional specific properties. This subgraph sometimes needs to be provided as a part of an input of an algorithm that constructs  $G'$ . We say that a reduction is a *linear-time reduction with respect to a subgraph  $K$*  if there exists a linear-time algorithm that transforms  $G$  to  $G'$  given  $G$  and a subgraph  $K$  with the specific properties. We will say that a reduction is a *linear-time reduction* if there exists a linear-time algorithm that both finds a suitable subgraph  $K$  and performs the reduction. If a graph  $G$  admits such a reduction, we will say that  $G$  has a linear-time reduction or that  $G$  has a linear-time reduction with respect to a subgraph  $K$ .

As an example of our reductions, we state and prove a lemma describing the simplest among our reductions; we leave the details of all other reductions because of the space constraints.

► **Lemma 7.** *Every non-basic 2-connected subcubic graph  $G$  that contains a cycle  $K$  with at most two vertices of degree three has a linear-time reduction.*

**Proof.** Since  $G$  is neither a cycle nor a  $\theta$ -graph, it follows that  $V(G) \neq V(K)$ . Since  $G$  is 2-connected,  $K$  contains exactly two vertices of degree three, say  $v_1$  and  $v_2$ . Let  $x_1$  and  $x_2$  be their neighbors outside of  $K$ , and let  $k_1$  and  $k_2$  be the the number of the internal vertices of the two paths between  $v_1$  and  $v_2$  in  $K$ . We can assume that  $k_1 \leq k_2$  by symmetry. If  $x_1 = x_2$ , then either  $G$  is a  $\theta$ -graph or  $x_1$  is incident with a cut-edge; since neither of these is possible, it holds that  $x_1 \neq x_2$ .

Suppose that  $k_1 = 0$  and  $k_2 = 1$ , i.e.,  $K$  is a triangle. Let  $z$  be the vertex of  $K$  distinct from  $v_1$  and  $v_2$ , and let  $G' = G - z$ . Note that  $G'$  is a 2-connected subcubic graph. We claim that  $G'$  is a reduction of  $G$ . Since  $n(G') = n(G) - 1$  and  $n_2(G') = n_2(G) + 1$ , it follows  $\delta(G, G') = 0$ . Consider a spanning Eulerian subgraph  $F'$  of  $G'$ . If  $F'$  contains the edge  $v_1v_2$ , then let  $F$  be the spanning Eulerian subgraph of  $G$  obtained from  $F'$  by removing the edge  $v_1v_2$  and adding the path  $v_1zv_2$ . If  $F'$  does not contain the edge  $v_1v_2$ , i.e.,  $v_1$  and  $v_2$  are isolated vertices of  $F'$ , then let  $F$  be the spanning Eulerian subgraph of  $G$  obtained from  $F'$  by adding the cycle  $K$ . It holds that  $\text{exc}(F) = \text{exc}(F')$  in both cases.

It remains to consider the case  $k_1 + k_2 \geq 2$ . Let  $G'$  be obtained from  $G - V(K)$  by adding a path  $x_1wx_2$  where  $w$  is a new vertex; note that  $w$  has degree two in  $G'$  and  $\delta(G, G') = 2(k_1 + k_2)$ . Since  $x_1 \neq x_2$ ,  $G'$  is simple. We show that  $G'$  is a reduction of  $G$ . Let  $F'$  be a spanning Eulerian subgraph of  $G'$ ; we will construct a spanning Eulerian subgraph

$F$  of  $G$ . If  $F'$  contains the path  $x_1wx_2$ , then let  $F$  be obtained from  $F' - w$  by adding the vertices of  $K$  and the edges  $x_1v_1, x_2v_2$ , and the path in  $K$  between  $v_1$  and  $v_2$  with  $k_2$  internal vertices. Note that the  $k_1$  vertices of the other path between  $v_1$  and  $v_2$  in  $K$  are isolated in  $F$ . Observe that

$$\text{exc}(F) = \text{exc}(F') + k_1 \leq \text{exc}(F') + \frac{k_1 + k_2}{2},$$

since  $k_1 \leq k_2$ . If  $w$  is an isolated vertex of  $F'$ , then let  $F$  be obtained from  $F' - w$  by adding the cycle  $K$ . In this case, we get that

$$\text{exc}(F) = \text{exc}(F') + 1 \leq \text{exc}(F') + \frac{k_1 + k_2}{2}.$$

Since it holds that  $\text{exc}(F) \leq \text{exc}(F') + \frac{1}{4}\delta(G, G')$  in both cases, the proof of the lemma is finished.  $\blacktriangleleft$

We next summarize the facts that can be established using our reduction techniques; the details are omitted because of the space constraints. We say that a 2-connected subcubic graph  $G$  is a *proper* graph if  $G$  is non-basic, has no cycle with at most four vertices of degree three, and has no cycle of length five or six with five vertices of degree three. We will call a non-basic 2-connected subcubic graph  $G$  *clean* if none of the lemmas that we have proven can be applied to  $G$ . Formally, a 2-connected subcubic graph  $G$  is *clean* if it is proper and

- (CT1) no cycle of length at most 7 in  $G$  contains a vertex of degree two,
- (CT2) every cycle of length six in  $G$  that is not a  $\theta$ -cycle is disjoint from all other cycles of length six,
- (CT3) every cycle  $K = v_1 \dots v_m$  of length  $m \leq 7$  in  $G$  satisfies that if each of the edges  $v_1v_m$  and  $v_2v_3$  is contained in a 2-edge-cut, then the edges  $v_1v_m$  and  $v_2v_3$  themselves form a 2-edge-cut, and
- (CT4) every cycle  $K = v_1 \dots v_6$  of length six in  $G$  satisfies at least one of the following
  - (a)  $K$  is a  $\theta$ -cycle, or
  - (b) each edge exiting  $K$  is contained in a 2-edge-cut but no two of them together form a 2-edge-cut, or
  - (c) each edge exiting  $K$  is contained in a 2-edge-cut, and there exists exactly one pair  $i$  and  $j$  with  $1 \leq i < j \leq 6$  such that the edges  $v_ix_i$  and  $v_jx_j$  form a 2-edge-cut, and this pair satisfies  $j - i = 3$ , or,
  - (d) precisely one edge exiting  $K$ , say  $v_1x_1$ , is not contained in a 2-edge-cut, and there exists a partition  $A$  and  $B$  of the vertices of  $G - V(K)$  such that  $x_1, x_2, x_6 \in A$ ,  $x_3, x_4, x_5 \in B$ , there is exactly one edge between  $A$  and  $B$ , and both  $A$  and  $B$  induce connected subgraphs of  $G - V(K)$ ,
 where  $x_i$  is the neighbor of the vertex  $v_i$  outside the cycle  $K$ ,  $i \in \{1, \dots, 6\}$ .

The next theorem summarizes our reduction results.

► **Theorem 8.** *There exists an algorithm running in time  $O(n^3)$  that constructs for a given  $n$ -vertex 2-connected subcubic graph  $G$  a reduction of  $G$  that is either basic or clean.*

## 4 Main result

We need few additional results before we can prove Theorem 1. The first concerns the structure of cycles passing through vertices of a cycle of length six in a clean 2-connected subcubic graph. Let  $v$  be a vertex of degree three in a graph  $G$ , and let  $x_1, x_2$  and  $x_3$  be its

neighbors. The *type* of  $v$  is the triple  $(\ell_1, \ell_2, \ell_3)$  such that  $\ell_1, \ell_2$  and  $\ell_3$  are the lengths of shortest cycles containing paths  $x_1vx_2, x_1vx_3$  and  $x_2vx_3$ . In our consideration, the order of the coordinates of the triple will be irrelevant, so we will always assume that the lengths satisfy that  $\ell_1 \leq \ell_2 \leq \ell_3$ . A type  $(\ell'_1, \ell'_2, \ell'_3)$  *dominates* the type  $(\ell_1, \ell_2, \ell_3)$  if  $\ell'_i \geq \ell_i$  for every  $i = 1, 2, 3$ . If  $K$  is a cycle in a graph  $G$  and each vertex of  $K$  has degree three, then the *type* of the cycle  $K$  is the multiset of the types of the vertices of  $K$ . Finally, a multiset  $M_1$  of types *dominates* a multiset  $M_2$  of types if there exists a bijection between the types contained in  $M_1$  and  $M_2$  such that each type of  $M_1$  dominates the corresponding type in  $M_2$ .

We can now show the following lemma (note that all vertices of the cycle  $K$  in the lemma must have degrees three since  $G$  is assumed to be clean). The proof is omitted because of the space constraints.

► **Lemma 9.** *Let  $G$  be a clean 2-connected subcubic graph and let  $K = v_1v_2 \dots v_6$  be a cycle of length six in  $G$ . If  $K$  is not a  $\theta$ -cycle, then the type of  $K$  dominates at least one of the following multisets:*

- $\{(6, 7, 7), (6, 7, 7), (6, 8, 8), (6, 8, 8), (6, 8, 8), (6, 8, 8)\}$ ,
- $\{(6, 7, 7), (6, 7, 8), (6, 7, 8), (6, 8, 8), (6, 8, 8), (6, 8, 8)\}$ , or
- $\{(6, 7, 7), (6, 7, 8), (6, 7, 9), (6, 7, 9), (6, 8, 8), (6, 8, 8)\}$ .

The following lemma follows from the description of the perfect matching polytope by Edmonds [6] and the fact that the perfect matching polytope has a strong separation oracle [17]; see e.g. [8] for further details.

► **Lemma 10.** *There exists a polynomial-time algorithm that for a given cubic 2-connected  $n$ -vertex graph outputs a collection of  $m \leq n/2 + 2$  perfect matchings  $M_1, \dots, M_m$  and non-negative coefficients  $a_1, \dots, a_m$  such that  $a_1 + \dots + a_m = 1$  and*

$$\sum_{i=1}^m a_i \chi_{M_i} = (1/3, \dots, 1/3) \in \mathbb{R}^{E(G)},$$

where  $\chi_{M_i} \in \mathbb{R}^{E(G)}$  is the characteristic vector of  $M_i$ .

Lemma 10 gives the following; we note that variants of Lemma 11 have also been used in [2, 7, 14].

► **Lemma 11.** *There exists a polynomial-time algorithm that for a given 2-connected  $n$ -vertex subcubic graph outputs a collection of  $m \leq n/2 + 2$  spanning Eulerian subgraphs  $F_1, \dots, F_m$  and probabilities  $p_1, \dots, p_m \geq 0, p_1 + \dots + p_m = 1$  that satisfy the following. If a spanning Eulerian subgraph  $F$  is equal to  $F_i$  with probability  $p_i, i = 1, \dots, m$ , then  $\mathbb{P}[e \in E(F)] = 2/3$ . In particular, a vertex of degree three is contained in a cycle of  $F$  with probability one and a vertex of degree two is isolated with probability  $1/3$ .*

We now combine Lemmas 9 and 11 to get the following.

► **Lemma 12.** *There exists a polynomial-time algorithm that given a clean 2-connected subcubic graph  $G$  outputs a spanning Eulerian subgraph  $F$  of  $G$  such that*

$$\text{exc}(F) \leq \frac{2n(G) + 2n_2(G)}{7}.$$

**Proof of Lemma 12.** We first apply the algorithm from Lemma 11 to get a collection of  $m \leq n/2 + 2$  spanning Eulerian subgraphs  $F_1, \dots, F_m$  and probabilities  $p_1, \dots, p_m$ . We show that

$$\mathbb{E}[\text{exc}(F)] \leq \frac{2n(G) + 2n_2(G)}{7}, \tag{2}$$



which implies the statement of the lemma since the number of the subgraphs  $F_1, \dots, F_m$  is linear in  $n$  and the excess of each them can be computed in linear time. In particular, the algorithm can output the subgraph  $F_i$  with the smallest  $\text{exc}(F_i)$ .

We now show that (2) holds. We apply a double counting argument, which we phrase as a discharging argument. At the beginning, we assign each vertex of degree three charge of  $2/7$  and to each vertex of degree two charge of  $4/7$ . Let  $c_1(v)$  be the initial charge of a vertex  $v$ . Note that the sum of the initial charges of the vertices is the right side of the inequality (2).

We next choose a random spanning Eulerian subgraphs  $F$  among the subgraphs  $F_1, \dots, F_m$  with probabilities given by  $p_1, \dots, p_m$ . The charge of each vertex that is isolated in  $F$  is decreased by one unit, and the charge of each vertex contained in a cycle of length  $k$  by  $2/k$  units. Let  $c_2(v)$  be the new charge of a vertex  $v$ . Observe that the total decrease of charge of the vertices is equal to  $\text{exc}(F)$ , i.e.,

$$\text{exc}(F) = \sum_{v \in V(G)} c_1(v) - c_2(v).$$

Hence, it is enough to prove that

$$\mathbb{E} \left[ \sum_{v \in V(G)} c_2(v) \right] \geq 0. \tag{3}$$

To prove (3), we consider the expectation of  $c_2(v)$  for individual vertices  $v$  of  $G$ .

If  $v$  is a vertex of  $G$  of degree two, then every cycle of  $G$  that contains  $v$  has length at least eight by (CT1). With probability  $1/3$ , the vertex  $v$  is isolated and loses one unit charge; with probability  $2/3$ , it is contained in a cycle and loses at most  $2/8 = 1/4$  units of charge. We conclude that

$$\mathbb{E}[c_2(v)] \geq \frac{4}{7} - \frac{1}{3} - \frac{2}{3} \cdot \frac{1}{4} = \frac{1}{14} > 0.$$

If  $v$  is a vertex of  $G$  of degree three with type  $(\ell_1, \ell_2, \ell_3)$ , we proceed as follows. Since each edge incident with  $v$  is contained in  $F$  with probability  $2/3$ ,  $v$  is contained in a cycle of  $F$  with a particular pair of its neighbors with probability  $1/3$ . It follows that the expected value of  $c_1(v)$  is at least

$$\mathbb{E}[c_2(v)] \geq \frac{2}{7} - \frac{1}{3} \left( \frac{2}{\ell_1} + \frac{2}{\ell_2} + \frac{2}{\ell_3} \right).$$

Since  $G$  is clean, the type of  $v$  dominates  $(6, 6, 6)$ . If the type of  $v$  dominates  $(7, 7, 7)$ , then  $\mathbb{E}[c_2(v)] \geq 0$ . Hence, we focus on vertices contained in cycles of length six in  $G$  in the rest of the proof.

Let  $K = v_1 \dots v_6$  be a cycle of length six in  $G$ . Since  $G$  is clean, each vertex of  $K$  has degree three. Suppose that  $K$  is not a  $\theta$ -cycle. By (CT2),  $K$  is disjoint from all other cycles of length six in  $G$ . Observe that

- if the type of  $v_i$  dominates  $(6, 8, 8)$ , then  $\mathbb{E}[c_2(v_i)] \geq \frac{1}{126}$ ,
- if the type of  $v_i$  dominates  $(6, 7, 7)$ , then  $\mathbb{E}[c_2(v_i)] \geq -\frac{1}{63}$ ,
- if the type of  $v_i$  dominates  $(6, 7, 8)$ , then  $\mathbb{E}[c_2(v_i)] \geq -\frac{1}{252}$ , and
- if the type of  $v_i$  dominates  $(6, 7, 9)$ , then  $\mathbb{E}[c_2(v_i)] \geq \frac{1}{189}$ .

Since the type of the cycle  $K$  dominates one of the three multisets listed in Lemma 9, it holds that

$$\mathbb{E}[c_2(v_1) + \dots + c_2(v_6)] \geq 0.$$

27:10 **Graphic TSP in Cubic Graphs**

It remains to analyze the case that  $K$  is a  $\theta$ -cycle. By symmetry, we can assume that the vertices  $v_1$  and  $v_4$  are its poles. Let  $x_i$  be the neighbor of  $v_i$  outside of  $K$ ,  $i = 1, \dots, 6$ . Further, let  $P = x_6v_6v_1v_2x_2$ ,  $P_1 = x_6v_6v_1$  and  $P_2 = x_2v_2v_1$ . Since each of the paths  $P_1$  and  $P_2$  is contained in  $F$  with probability  $1/3$ , the subgraph  $F$  contains the path  $P$  with probability at most  $1/3$ ; let  $p$  be this probability. Since  $G$  is clean (and so proper), the distance between  $x_2$  and  $x_3$  in  $G - V(K)$  is at least three; likewise, the distance between  $x_5$  and  $x_6$  in  $G - V(K)$  is at least three. Hence, any cycle containing  $P_1$  or  $P_2$  has length at least 10, and any cycle containing  $P$  has length at least 14. Since  $F$  contains the path  $P$  with probability  $p$ , the path  $P_1$  but not  $P$  with probability  $1/3 - p$ , the path  $P_2$  but not  $P$  with probability  $1/3 - p$ , and neither  $P_1$  nor  $P_2$  with probability  $1/3 + p$ , it follows that

$$\mathbb{E}[c_2(v_1)] = \frac{2}{7} - p \cdot \frac{1}{7} - 2 \left( \frac{1}{3} - p \right) \cdot \frac{1}{5} - \left( \frac{1}{3} + p \right) \cdot \frac{1}{3} = \frac{13}{315} - \frac{8}{105}p \geq \frac{1}{63}.$$

The symmetric argument yields that  $\mathbb{E}[c_2(v_4)] \geq \frac{1}{63}$ . Since every cycle in  $G$  containing the path  $P_2$  has length at least 10, the type of  $v_2$  dominates  $(6, 6, 10)$  and thus  $\mathbb{E}[c_2(v_2)] \geq -\frac{1}{315}$ . The same holds for vertices  $v_3, v_5$  and  $v_6$ .

Let  $Q_1$  be the set of all poles of  $\theta$ -cycles in  $G$ , and let  $Q_2$  be the set of vertices contained in  $\theta$ -cycle that are not a pole of a (possibly different)  $\theta$ -cycle. Since each vertex of  $Q_2$  has a neighbor in  $Q_1$ , it follows  $|Q_2| \leq 3|Q_1|$ . The previous analysis yields that

$$\mathbb{E} \left[ \sum_{v \in Q_1 \cup Q_2} c_2(v) \right] \geq |Q_1| \left( \frac{1}{63} - 3 \cdot \frac{1}{315} \right) = \frac{2}{315}|Q_1| \geq 0.$$

Since the set  $Q_1 \cup Q_2$  and the vertex set of cycles of length six that are not  $\theta$ -cycles are disjoint, the inequality (3) follows.  $\blacktriangleleft$

We are ready to prove Theorems 1 and 2.

**Proof of Theorem 1.** By Observation 6, it is enough to construct a spanning Eulerian subgraph  $F$  of  $G$  with

$$\text{exc}(F) \leq \frac{2(n(G) + n_2(G))}{7} + 1.$$

If  $G$  is basic, such a subgraph  $F$  exists by Observation 5, and can easily be constructed in polynomial time. If  $G$  is not basic, we can find a reduction  $G'$  of  $G$  that is either basic or clean in polynomial time by Theorem 8.

If  $G'$  is basic, then we find a spanning Eulerian subgraph with

$$\text{exc}(F') \leq \frac{2(n(G') + n_2(G'))}{7} + 1$$

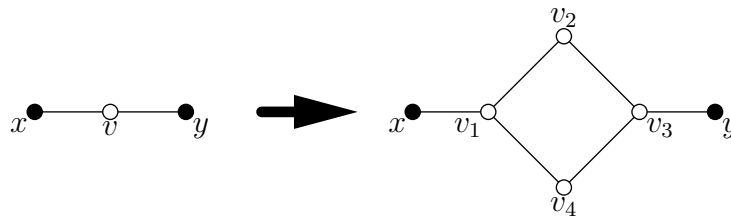
as in the case when  $G$  itself is basic. If  $G'$  is clean, then Lemma 12 yields that we can construct in polynomial time a spanning Eulerian subgraph  $F'$  of  $G'$  such that

$$\text{exc}(F') \leq \frac{2(n(G') + n_2(G'))}{7}.$$

Since  $G'$  is a reduction of  $G$ , we can find in polynomial time a spanning Eulerian subgraph  $F$  of  $G$  such that

$$\text{exc}(F) \leq \text{exc}(F') + \frac{\delta(G, G')}{4} \leq \text{exc}(F') + \frac{2\delta(G, G')}{7} \leq \frac{2(n(G') + n_2(G'))}{7} + 1,$$

which finishes the proof of the theorem.  $\blacktriangleleft$



■ **Figure 2** Replacing a vertex of degree two with a cycle of length four in Lemma 13.

**Proof of Theorem 2.** Let  $G$  be an input cubic graph and let  $n$  be the number of its vertices. We assume that  $G$  is connected since  $G$  would not have a TSP walk otherwise. Let  $F$  be the set of bridges of  $G$ , which can be found in linear time using the standard algorithm based on DFS. Further, let  $G'$  be the graph obtained from  $G$  by removing the edges of  $F$ , and let  $n_0$  and  $n_2$  be the number of its vertices of degree zero and two, respectively. Note that  $G'$  has no vertices of degree one since if two edges incident with a vertex  $v$  in a cubic graph are bridges, then the third edge incident with  $v$  is also a bridge. Finally, let  $k$  be the number of non-trivial components of  $G'$ , i.e., the components of  $G'$  that are not formed by a single vertex. Observe that the number of vertices of degree two in  $G'$  is at most  $2k - 2$ , i.e.,  $n_2 \leq 2k - 2$ .

We next apply the algorithm from Theorem 1 to each non-trivial component of  $G'$ , and obtain a collection of  $k$  TSP walks such that the sum of their lengths is at most

$$\frac{9}{7}(n - n_0) + \frac{2}{7}n_2 - k.$$

These  $k$  TSP walks can be connected by traversing each of the edges of  $F$  twice, which yields a TSP walk in  $G$  of total length at most

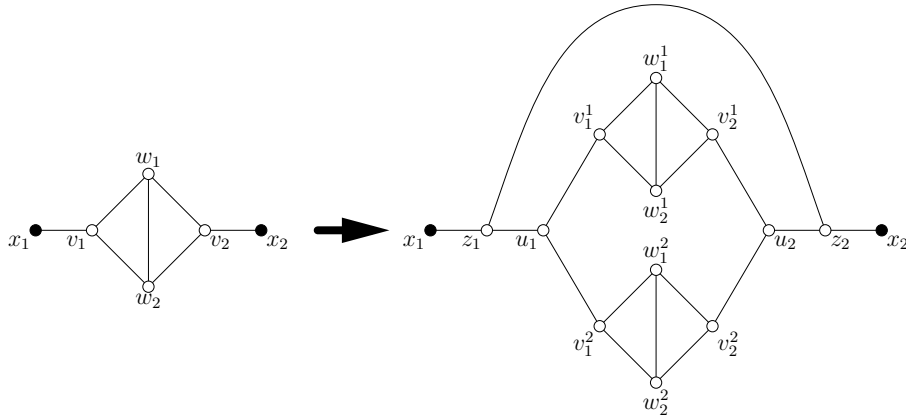
$$\frac{9}{7}(n - n_0) + \frac{2}{7}n_2 - k + 2|F| \leq \frac{9}{7}(n - n_0) + 2|F|. \quad (4)$$

The inequality in (4) follows from the inequality  $n_2 \leq 2k - 2$ , which we have observed earlier in the proof. Since any TSP walk in  $G$  must have length at least  $(n - n_0) + 2|F|$  (it must contain at least  $n - n_0$  edges inside the non-trivial blocks and each bridge must be traversed twice), the upper bound in (4) on the length of the constructed TSP walk is at most the multiple of  $9/7$  of the length of the optimal TSP walk in  $G$ , which yields the desired approximation factor of the algorithm. ◀

## 5 Lower bounds

In this section, we provide two constructions of 2-connected subcubic graphs that illustrate that the bound claimed in Conjecture 3 would be the best possible. The constructions are based on two operations that we analyze in Lemmas 13 and 15. The proofs of the lemmas are omitted because of the space constraints.

► **Lemma 13.** *Let  $G$  be a 2-connected subcubic graph, let  $v$  be a vertex of  $G$  that has exactly two neighbors, and let  $x$  and  $y$  be its two neighbors. Further, let  $G'$  be the graph obtained from  $G$  by removing the vertex  $v$ , adding a cycle  $v_1v_2v_3v_4$  and edges  $xv_1$  and  $yv_3$  as in Figure 2. The graph  $G'$  is a 2-connected subcubic graph and it holds that  $n(G') = n(G) + 3$ ,  $n_2(G') = n_2(G) + 1$  and  $\text{minexc}(G') = \text{minexc}(G) + 1$ .*



■ **Figure 3** The operation of replacing a diamond analyzed in Lemma 15.

Repeated applications of the operation described in Lemma 13 starting with the graph  $K_{2,3}$  yields the following.

► **Proposition 14.** *For every integer  $n \geq 5$ ,  $n \equiv 2 \pmod 3$ , there exists a 2-connected subcubic  $n$ -vertex graph  $G$  such that*

$$\text{minexc}(G) = \frac{n(G) + n_2(G)}{4} + 1.$$

The second operation is more involved. A *diamond* in a graph  $G$  is an induced subgraph isomorphic to  $K_4^-$ , i.e., the graph  $K_4$  with one edge removed.

► **Lemma 15.** *Let  $G$  be a 2-connected cubic graph containing a diamond  $D$ . Let  $v_1, v_2, w_1$  and  $w_2$  be the vertices of the diamond as depicted in Figure 3, and let  $x_1$  and  $x_2$  be the neighbors of  $v_1$  and  $v_2$  outside of the diamond  $D$ . Further, let  $G'$  be the graph obtained from  $G$  by removing the vertices of the diamond  $D$  and inserting the subgraph depicted in Figure 3. The graph  $G'$  is a 2-connected cubic graph with  $n(G') = n(G) + 8$  and  $\text{minexc}(G') = \text{minexc}(G) + 2$ . Moreover, the graph  $G'$  contains at least two diamonds.*

Consider the cubic graph formed by two diamonds and two edges joining the vertices of degree two in different diamonds, and repeatedly apply the operation described in Lemma 15.

► **Proposition 16.** *For every integer  $n \geq 8$ ,  $n \equiv 0 \pmod 8$ , there exists a 2-connected cubic  $n$ -vertex graph  $G$  with  $\text{minexc}(G) = n/4$ .*

Propositions 14 and 16, and Observation 6 yield that neither the coefficient  $5/4$  nor the coefficient  $1/4$  in Conjecture 3 can be improved. Indeed, for every  $\alpha < 5/4$ , there exist infinitely many 2-connected cubic graphs  $G$  with  $\text{tsp}(G) > \alpha n(G) + o(n(G))$  by Proposition 16. Likewise, for every  $\beta < 1/4$ , there exist infinitely many 2-connected subcubic graphs  $G$  with  $\text{tsp}(G) > \frac{5}{4}n(G) + \beta n_2(G) + o(n(G))$ . While neither of the two coefficients in Conjecture 3 can be improved, it may be possible to prove a stronger bound that is not linear in both  $n(G)$  and  $n_2(G)$ .

**Acknowledgements.** The authors would like to thank the anonymous reviewers for their valuable and insightful comments on the submission.

---

**References**

---

- 1 Nishita Aggarwal, Naveen Garg, and Swati Gupta. A  $4/3$ -approximation for TSP on cubic 3-edge-connected graphs. *arXiv e-prints*, 1101.5586, 2011.
- 2 Sylvia Boyd, René Sitters, Suzanne van der Ster, and Leen Stougie. The traveling salesman problem on cubic and subcubic graphs. *Mathematical Programming*, 144(1-2):227–245, 2014.
- 3 Barbora Candráková and Robert Lukot'ka. Cubic TSP – a 1.3-approximation. *arXiv e-prints*, 1506.06369, 2015.
- 4 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, DTIC Document, 1976.
- 5 José Correa, Omar Larré, and José A Soto. TSP tours in cubic graphs: beyond  $4/3$ . *SIAM Journal on Discrete Mathematics*, 29(2):915–939, 2015.
- 6 Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *J. Res. Nat. Bur. Standards B*, 69(1965):125–130, 1965.
- 7 David Gamarnik, Moshe Lewenstein, and Maxim Sviridenko. An improved upper bound for the TSP in cubic 3-edge-connected graph. *Operations Research Letters*, 33:467–474, 2005.
- 8 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2. Springer Science & Business Media, 2012.
- 9 Jeremy Karp and R Ravi. A  $9/7$ -approximation algorithm for graphic TSP in cubic bipartite graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*, volume 28 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 284–296. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.284.
- 10 Marek Karpinski, Michael Lampis, and Richard Schmied. New inapproximability bounds for TSP. *Journal of Computer and System Sciences*, 81:1665–1677, 2015.
- 11 Marek Karpinski and Richard Schmied. Approximation hardness of graphic TSP on cubic graphs. *RAIRO Operations Research*, 49:651–668, 2015.
- 12 M. Lampis. Improved inapproximability for TSP. *Lecture Notes in Computer Science*, 7408:243–253, 2013.
- 13 Jan Mazák and Robert Lukot'ka. Simple cubic graphs with no short travelling salesman tour, 2016. Manuscript.
- 14 Tobias Mömke and Ola Svensson. Approximating graphic TSP by matchings. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 560–569. IEEE, 2011.
- 15 Marcin Mucha.  $13/9$ -approximation for graphic TSP. *Theory of Computing Systems*, 55:640–657, 2014.
- 16 Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the traveling salesman problem. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 550–559. IEEE, 2011.
- 17 Manfred W. Padberg and M. R. Rao. Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, 7(1):67–80, 1982.
- 18 Christos H. Papadimitriou and Santosh Vempala. On the approximability of the traveling salesman problem. *Combinatorica*, 26:101–120, 2006.
- 19 András Sebő and Jens Vygen. Shorter tours by nicer ears:  $7/5$ -approximation for the graph-TSP,  $3/2$  for the path version, and  $4/3$  for two-edge-connected subgraphs. *Combinatorica*, 34:597–629, 2014.
- 20 Anke van Zuylen. Improved approximations for cubic and cubic bipartite TSP. In *Integer Programming and Combinatorial Optimization (IPCO)*, volume 9682 of *Lecture Notes in Computer Science*, pages 250–261. Springer, 2016.



# Independent Sets near the Lower Bound in Bounded Degree Graphs\*

Zdeněk Dvořák<sup>1</sup> and Bernard Lidický<sup>2</sup>

- 1 Charles University, Prague, Czech Republic  
rakdver@iuuk.mff.cuni.cz
- 2 Iowa State University, Ames, IA, USA  
lidicky@iasate.edu

---

## Abstract

By Brook's Theorem, every  $n$ -vertex graph of maximum degree at most  $\Delta \geq 3$  and clique number at most  $\Delta$  is  $\Delta$ -colorable, and thus it has an independent set of size at least  $n/\Delta$ . We give an approximate characterization of graphs with independence number close to this bound, and use it to show that the problem of deciding whether such a graph has an independent set of size at least  $n/\Delta + k$  has a kernel of size  $O(k)$ .

**1998 ACM Subject Classification** G.2.2 Graph Theory, F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** independent set, bounded degree,  $\Delta$ -colorable, fixed parameter tractability

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.28

## 1 Introduction

Let  $\Delta \geq 3$  be an integer and let  $G$  be an  $n$ -vertex graph of maximum degree at most  $\Delta$  that does not contain a clique of size  $\Delta + 1$ . By Brooks' Theorem [6]  $G$  is  $\Delta$ -colorable, and thus the size of the largest independent set in  $G$  (which we denote by  $\alpha(G)$ ) is at least  $n/\Delta$ . This bound is tight, as evidenced by graphs obtained from a disjoint union of cliques of size  $\Delta$  by adding a matching. However, if the bound on the size of the largest clique (which we denote by  $\omega(G)$ ) is tightened, further improvements are possible.

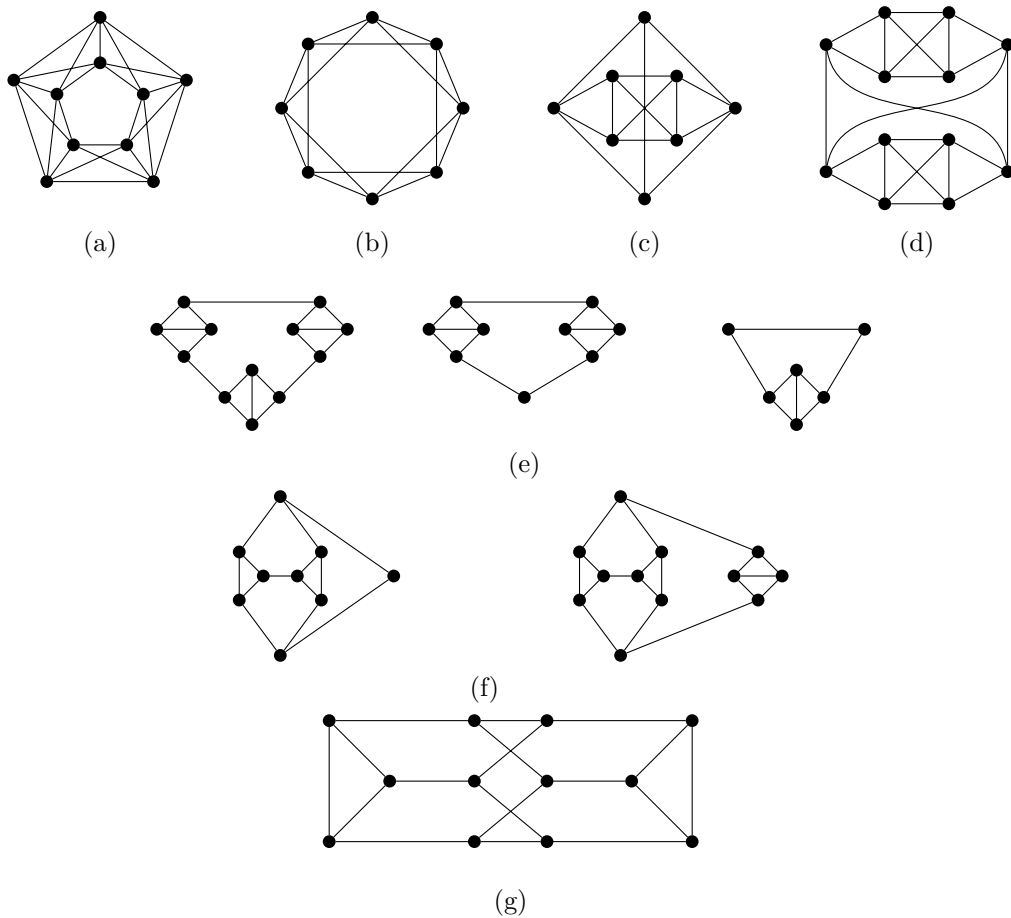
Albertson, Bollobás and Tucker [1] proved that if a connected graph  $G$  of maximum degree  $\Delta$  satisfies  $\omega(G) \leq \Delta - 1$ , then  $\alpha(G) > n/\Delta$ , unless  $\Delta = 4$  and  $G = C_8^2$  or  $\Delta = 5$  and  $G = C_5 \boxtimes K_2$  (see Figure 1(a) and (b)). Under the same assumptions, the bound was further strengthened by King, Lu and Peng [18], who proved that  $G$  has fractional chromatic number at most  $\Delta - \frac{2}{67}$ , and thus  $\alpha(G) \geq \frac{n}{\Delta - \frac{2}{67}}$ ; for particular values of  $\Delta$ , this bound has been further improved in [22, 16, 11, 12], and Fajtlowicz [14] gave better bounds for graphs with smaller clique number.

In this paper, we study the case when  $\omega(G) \leq \Delta$  in more detail. How do graphs with largest independent set close to  $n/\Delta$  look like? An infinite class of examples can be obtained by generalizing the construction from the first paragraph: if  $G$  contains a subgraph  $H$  isomorphic to the disjoint union of cliques of size  $\Delta$  and  $|V(G) \setminus V(H)| \leq k$ , then clearly  $\alpha(G) \leq |V(H)|/\Delta + |V(G) \setminus V(H)| \leq n/\Delta + k$ . For  $\Delta \geq 6$ , we prove an approximate converse

---

\* This work was partially supported by (FP7/2007-2013)/ERC Consolidator grant LBCAD no. 616787 and NSF grant DMS-1600390.





■ **Figure 1** Non-clique  $\Delta$ -tight graphs.

to this statement: If  $\max(\Delta(G), \omega(G)) \leq \Delta$  and  $\alpha(G) < n/\Delta + k$ , then  $G$  contains a subgraph  $H$  isomorphic to the disjoint union of cliques of size  $\Delta$  such that  $|V(G) \setminus V(H)| < 34\Delta^2 k$ .

For  $\Delta \in \{3, 4, 5\}$ , a more technical statement is needed. We say that a graph  $H$  is  $\Delta$ -tight if

- $H$  is a clique on  $\Delta$  vertices; or
- $\Delta = 5$  and  $H = C_5 \boxtimes K_2$  (Figure 1(a)); or
- $\Delta = 4$  and  $H = C_8^2$  (Figure 1(b)); or
- $\Delta = 4$  and  $H$  is the graph depicted in Figure 1(c), which we call an *extended clique*, and its vertices of degree 3 are its *attachments*; or,
- $\Delta = 4$  and  $H$  is the graph depicted in Figure 1(d), which we call an *extended double-clique*;  
or
- $\Delta = 3$  and  $H$  is one of the graphs depicted in Figure 1(e), which we call *diamond necklaces*;  
or
- $\Delta = 3$  and  $H$  is one of the graphs depicted in Figure 1(f), which we call *Havel necklaces*;  
or,
- $\Delta = 3$  and  $H$  is the graph depicted in Figure 1(g), which we call a *triangle-dominated 6-cycle*.

Observe that if  $H$  is  $\Delta$ -tight, then  $\alpha(H) = |V(H)|/\Delta$ . We say that a graph  $G_0$  is  $\Delta$ -tightly partitioned if there exists a partition of the vertices of  $G_0$  such that each part induces a  $\Delta$ -



tight subgraph of  $G_0$ . Clearly, a  $\Delta$ -tightly partitioned graph  $G_0$  satisfies  $\alpha(G_0) \leq |V(G_0)|/\Delta$ . Our main result now can be stated as follows.

► **Theorem 1.** *Let  $\Delta \geq 3$  and  $k \geq 0$  be integers, and let  $G$  be an  $n$ -vertex graph with  $\max(\Delta(G), \omega(G)) \leq \Delta$ . If  $\alpha(G) < n/\Delta + k$ , then there exists a set  $X \subseteq V(G)$  of size less than  $34\Delta^2 k$  such that  $G - X$  is  $\Delta$ -tightly partitioned.*

Note that the set  $X$  of Theorem 1 can be found in time  $O(\Delta^2 n)$  without specifying  $k$  in advance (cf. a stronger statement in Lemma 11 below), and that every graph containing such a set  $X$  satisfies  $\alpha(G) \leq |V(G)|/\Delta + |X|$ . Hence, we obtain the following algorithmic consequence.

► **Corollary 2.** *Let  $\Delta \geq 3$  be an integer, and let  $G$  be an  $n$ -vertex graph with  $\max(\Delta(G), \omega(G)) \leq \Delta$ . The difference  $\alpha(G) - n/\Delta$  can be approximated up to factor  $34\Delta^2$  in time  $O(\Delta^2 n)$ .*

Another application of Theorem 1 (or more precisely, its refinement Lemma 11) concerns fixed-parameter tractability. An algorithmic problem is called *fixed-parameter tractable* with respect to a parameter  $p$  if there exists a computable function  $f$ , a polynomial  $q$ , and an algorithm that solves each input instance  $Z$  in time  $f(p(Z))q(|Z|)$ . This notion has been influential in the area of computational complexity, giving a plausible approach towards many otherwise intractable problems [8, 20, 7].

A popular choice of the parameter is the value of the solution; i.e., such fixed-parameter tractability results show that the solution to the problem can be found quickly if its value is small. However, in the case of the problem of finding the largest independent set when restricted to some class of sparse graphs, this parameterization makes little sense – the problem is fixed-parameter tractable for the trivial (and unhelpful) reason that all large graphs in the class have large independent sets. In this setting, parameterization by the excess of the size of the largest independent set over the lower bound for the independence number in the class is more reasonable.

Let  $\mathcal{G}$  be a class of graphs, and let  $f(n) = \min\{\alpha(G) : G \in \mathcal{G}, |V(G)| = n\}$ . Let  $(\mathcal{G}, \alpha)$ -ATLB (Above Tight Lower Bound) denote the algorithmic problem of deciding, for an  $n$ -vertex graph  $G \in \mathcal{G}$  and an integer  $k \geq 0$ , whether  $\alpha(G) \geq f(n) + k$ . For specific graph classes  $\mathcal{G}$ , we are interested in the fixed-parameter tractability of this problem when parameterized by  $k$ , i.e., in finding algorithms for this problem with time complexity  $f(k)\text{poly}(n)$ .

The best-known case of this problem concerns the class  $\mathcal{P}$  of planar graphs. By the Four Color Theorem [2, 3], all  $n$ -vertex planar graphs have independent sets of size at least  $\lceil n/4 \rceil$ , and this lower bound is tight. However, there is not even a polynomial-time algorithm known to decide whether  $\alpha(G) > n/4$  for an  $n$ -vertex planar graph  $G$ , and consequently the complexity of  $(\mathcal{P}, \alpha)$ -ATLB is wide open [20, 4, 15]. The variant of the problem for planar triangle-free graphs was solved by Dvořák and Mnich [9]. Relevantly to the current paper, they also considered the case of planar graphs of maximum degree 4 and showed it to be fixed-parameter tractable [19, 10]; for both problems, they obtained algorithms with time complexity  $2^{O(\sqrt{k})}n$ .

As a consequence of our theory, we strengthen the last mentioned result in multiple directions. Firstly, we can drop the assumption of planarity and relax the condition on the maximum degree to an arbitrary integer  $\Delta \geq 3$  (assuming that cliques of size  $\Delta + 1$  are forbidden – let  $\mathcal{G}_\Delta$  denote the class of such graphs). Secondly, we obtain a *linear-size kernel* for the problem, i.e., we show that in polynomial time (specifically,  $O(\Delta^2 n)$ ), each instance of the problem can be reduced to an instance of size  $O(\Delta^3 k)$ .

► **Corollary 3.** *There exists an algorithm with time complexity  $O(\Delta^2 n)$  that, given as an input an integer  $\Delta \geq 3$ , an  $n$ -vertex graph  $G$  with  $\max(\Delta(G), \omega(G)) \leq \Delta$ , and an integer  $k \geq 0$ , returns an induced subgraph  $G_0$  of  $G$  with  $n_0 \leq 114\Delta^3 k$  vertices such that  $\alpha(G) \geq n/\Delta + k$  if and only if  $\alpha(G_0) \geq n_0/\Delta + k$ .*

Such an instance can be solved by brute force, leading to a  $2^{O(\Delta^3 k)} + O(\Delta^2 n)$  algorithm for  $(\mathcal{G}_\Delta, \alpha)$ -ATLB. Furthermore, as Robertson et al. [21] showed, an  $m$ -vertex planar graph has tree-width  $O(\sqrt{m})$ , and thus its largest independent set can be found in time  $2^{O(\sqrt{m})}$ . Hence, for the case of planar graphs of maximum degree 4 previously considered in [19, 10], we can improve the time complexity to  $2^{O(\sqrt{k})} + O(n)$ .

After introducing graph-theoretic results on sizes of independent sets in Section 2, we prove Theorem 1 and Corollaries 2 and 3 in Section 3.

## 2 Many vertices not covered by $\Delta$ -tight subgraphs

We say that a vertex  $v \in V(G)$  is  $\Delta$ -free if  $v$  is not contained in any  $\Delta$ -tight induced subgraph of  $G$ . As a special case, Theorem 1 implies that if  $G$  has many  $\Delta$ -free vertices, then its largest independent set is much larger than the lower bound  $n/\Delta$ . In this section, we prove this special case in Lemmas 7 and 9. Let us start by a simple observation.

► **Lemma 4.** *Let  $\Delta > 0$  be an integer and let  $G$  be a graph with  $\Delta(G) \leq \Delta$ , and let  $K$  be a clique of size  $\Delta$  in  $G$ . Consider one of the following situations:*

- *$K$  contains a vertex  $v$  whose degree in  $G$  is  $\Delta - 1$ , and  $G_0 = G - V(K)$ , or*
- *$V(G) \setminus V(K)$  contains two adjacent vertices  $u'$  and  $v'$  with neighbors in  $K$ , and  $G_0 = G - V(K)$ , or*
- *$V(G) \setminus V(K)$  contains two distinct non-adjacent vertices  $u'$  and  $v'$  with neighbors in  $K$ , and  $G_0 = G - V(K) + u'v'$ .*

*Then  $\alpha(G) \geq \alpha(G_0) + 1$ .*

**Proof.** Since  $\Delta(G) \leq \Delta$ , every vertex of  $K$  has at most one neighbor not in  $K$ . In the last two cases, let  $u$  and  $v$  be the neighbors of  $u'$  and  $v'$  in  $K$ , respectively. Consider an independent set  $S$  of  $G_0$  of size  $\alpha(G_0)$ .

In the first case,  $v$  has no neighbor in  $S$ , as all neighbors of  $v$  belong to  $K$ . In the last two cases, since  $u'v' \in E(G_0)$ , we can by symmetry assume that  $v' \notin S$ . Hence,  $S \cup \{v\}$  is in both cases an independent set in  $G$ . ◀

We say that distinct vertices  $u$  and  $v$  are  $\Delta$ -adjacent in a graph  $G$  if  $uv \notin E(G)$  and  $G$  contains an induced subgraph  $H_0$  with  $u, v \in V(H_0)$  such that  $H_0 + uv$  is  $\Delta$ -tight.

For a vertex  $v$ , let  $N_G[v]$  denote the set consisting of  $v$  and of the neighbors of  $v$  in  $G$ , and for a set  $S \subseteq V(G)$ , let  $N_G[S] = \bigcup_{v \in S} N_G[v]$ . We say that a set  $Z \subseteq V(G)$  is  $\Delta$ -profitable if  $G[Z]$  contains an independent set  $S$  such that  $N_G[S] \subseteq Z$  and  $|Z| \leq \Delta|S| - 1$ . Thus, any independent set of  $G - Z$  can be extended to an independent set in  $G$  by adding  $S$ , and this way the size of the independent set increases by slightly more than  $|Z|/\Delta$ .

► **Lemma 5.** *Let  $\Delta \geq 4$  be an integer and let  $G$  be a graph with  $\Delta(G) \leq \Delta$ . Let  $S = \{v_1, v_2, v_3\}$  be a subset of vertices of  $G$ . If  $G$  contains no  $\Delta$ -profitable set of size at most  $\Delta + 10$ , then the vertices of  $S$  are not pairwise  $\Delta$ -adjacent.*

**Proof.** Suppose for a contradiction that vertices of  $S$  are pairwise  $\Delta$ -adjacent, and in particular  $S$  is an independent set. For  $1 \leq i < j \leq 3$ , let  $H_{ij}$  be the induced subgraph of  $G$  showing  $\Delta$ -adjacency of  $v_i$  and  $v_j$ .

Suppose first that all integers  $i$  and  $j$  such that  $1 \leq i < j \leq 3$  satisfy  $|N_G[v_i] \cap N_G[v_j]| \geq \Delta - 2$ . Let  $Z = N_G[S]$ ; by the principle of inclusion and exclusion we have

$$\begin{aligned} |Z| &= \sum_{i=1}^3 |N_G[v_i]| - \sum_{1 \leq i < j \leq 3} |N_G[v_i] \cap N_G[v_j]| + \left| \bigcap_{i=1}^3 N_G[v_i] \right| \\ &\leq \sum_{i=1}^3 |N_G[v_i]| - \sum_{i=2}^3 |N_G[v_1] \cap N_G[v_i]| \\ &\leq 3(\Delta + 1) - 2(\Delta - 2) = \Delta + 7 \leq 3\Delta - 1 = \Delta|S| - 1. \end{aligned}$$

It follows that  $Z$  is  $\Delta$ -profitable, which is a contradiction.

Hence, we can assume that say  $|N_G[v_1] \cap N_G[v_2]| \leq \Delta - 3$ , and in particular  $H'_{12} = H_{12} + v_1v_2$  is not a clique of size  $\Delta$ . Consequently,  $\Delta \in \{4, 5\}$ . Note that  $H'_{12}$  is not an extended clique with attachments  $v_1$  and  $v_2$ , as otherwise we would have  $|N_G[v_1] \cap N_G[v_2]| = 2 = \Delta - 2$ .

If  $H'_{12}$  is not an extended double-clique, then  $H'_{12}$  is  $C_8^2$ ,  $C_5 \boxtimes K_2$ , or the extended clique, and  $|V(H_{12})| = 2\Delta$ . A vertex of  $V(H_{12})$  may only have neighbors outside of  $V(H_{12})$  in  $G$  if its degree in  $H'_{12}$  is less than  $\Delta$ , or if it is one of  $v_1$  and  $v_2$ . Using this observation, a case analysis shows that  $H_{12}$  contains an independent set  $S'$  of size three including  $v_1$  and  $v_2$ , such that  $|N_G[S'] \setminus V(H_{12})| \leq 7 - \Delta$ . Letting  $Z' = N_G[S'] \cup V(H_{12})$ , we have  $|Z'| \leq 2\Delta + (7 - \Delta) = \Delta + 7 \leq 3\Delta - 1 = \Delta|S'| - 1$ . Hence,  $Z'$  is  $\Delta$ -profitable, which is a contradiction.

If  $H'_{12}$  is an extended double-clique (and  $\Delta = 4$ ), then  $|V(H_{12})| = 12$  and  $H_{12}$  contains an independent set  $S'$  of size 4 (including  $v_1$  and  $v_2$ ) such that  $|N_G[S'] \setminus V(H_{12})| \leq 2$ . Letting  $Z' = N_G[S'] \cup V(H_{12})$ , we have  $|Z'| \leq 14 = \Delta + 10 < 4\Delta - 1 = \Delta|S'| - 1$ . Again,  $Z'$  is  $\Delta$ -profitable, which is a contradiction.  $\blacktriangleleft$

King, Lu and Peng [18] proved that every graph  $G$  of maximum degree at most  $\Delta \geq 4$  and with no  $\Delta$ -tight induced subgraph has fractional chromatic number at most  $\Delta - 2/67$ . This gives a lower bound on the independence number of graphs whose vertices are all  $\Delta$ -free.

**► Corollary 6.** *Let  $\Delta \geq 4$  be an integer, and let  $G$  be an  $n$ -vertex graph of maximum degree at most  $\Delta$ . If  $G$  contains no  $\Delta$ -tight induced subgraph, then  $\alpha(G) \geq \frac{n}{\Delta} + \frac{1}{34\Delta^2}n$ .*

**Proof.** Since  $G$  has fractional chromatic number at most  $\frac{67\Delta-2}{67}$ , we have

$$\begin{aligned} \alpha(G) &\geq \frac{67n}{67\Delta - 2} = \frac{(67 - 2/\Delta)n + 2n/\Delta}{67\Delta - 2} \\ &= \frac{n}{\Delta} + \frac{2n}{\Delta(67\Delta - 2)} \geq \frac{n}{\Delta} + \frac{n}{34\Delta^2}. \end{aligned} \quad \blacktriangleleft$$

We now extend this result to graphs with only some  $\Delta$ -free vertices.

**► Lemma 7.** *Let  $\Delta \geq 4$  be an integer, and let  $G$  be an  $n$ -vertex graph with  $\max(\Delta(G), \omega(G)) \leq \Delta$ . If  $G$  contains at least  $m$   $\Delta$ -free vertices, then  $\alpha(G) \geq \frac{n}{\Delta} + \frac{1}{34\Delta^2}m$ .*

**Proof.** We proceed by induction, and thus we can assume that the claim holds for all graphs with less than  $n$  vertices.

If  $H$  is a  $\Delta$ -tight induced subgraph of  $G$  that is not a clique, then observe that  $\alpha(H) = t$  for some  $t \in \{2, 3\}$  and that  $H$  has an independent set of size  $t$  whose closed neighborhood in  $G$  is contained in  $V(H)$ . Consequently  $\alpha(G) = \alpha(G - V(H)) + t$ . Furthermore,  $|V(H)| = t\Delta$  and  $G - V(H)$  has at least  $m$   $\Delta$ -free vertices, and thus  $\alpha(G) = \alpha(G - V(H)) + t \geq$

$\frac{n-t\Delta}{\Delta} + \frac{1}{34\Delta^2}m + t = \frac{n}{\Delta} + \frac{1}{34\Delta^2}m$  by the induction hypothesis. Hence, we can without loss of generality assume that the only  $\Delta$ -tight induced subgraphs of  $G$  are cliques of size  $\Delta$ .

Suppose that  $G$  contains a  $\Delta$ -profitable set  $Z$  of size at most  $\Delta + 10$ , and let  $S$  be the corresponding independent set. Note that the number of  $\Delta$ -free vertices of  $G - Z$  is at least  $m - |Z|$ , and thus

$$\begin{aligned} \alpha(G) &\geq \alpha(G - Z) + |S| \\ &\geq \frac{n - |Z|}{\Delta} + \frac{1}{34\Delta^2}(m - |Z|) + |S| \\ &= \frac{n}{\Delta} + \frac{1}{34\Delta^2}m + \frac{\Delta|S| - |Z| - |Z|/(34\Delta)}{\Delta}. \end{aligned}$$

Since  $Z$  is  $\Delta$ -profitable, we have  $\Delta|S| - |Z| \geq 1 > \frac{|Z|}{34\Delta}$ , and thus the inequality we seek holds.

If  $\omega(G) < \Delta$ , then all vertices of  $G$  are  $\Delta$ -free and the claim follows from Corollary 6. It remains to consider the case that  $G$  contains a clique  $K$  of size  $\Delta$ , but does not contain any  $\Delta$ -profitable sets with at most  $\Delta + 10$  vertices. Let  $S$  be the set of vertices outside  $K$  with a neighbor in  $K$ . If  $K$  contains a vertex whose degree in  $G$  is  $\Delta - 1$ , or if  $S$  is not an independent set, then note that  $G - V(K)$  contains at least  $m$   $\Delta$ -free vertices, and by Lemma 4 and the induction hypothesis, we have  $\alpha(G) \geq \alpha(G - V(K)) + 1 \geq \frac{n-\Delta}{\Delta} + \frac{1}{34\Delta^2}m + 1 = \frac{n}{\Delta} + \frac{1}{34\Delta^2}m$ . Hence, suppose that  $S$  is an independent set and each vertex of  $K$  has a neighbor in  $S$  (and thus there are precisely  $\Delta$  edges between  $V(K)$  and  $S$ ).

Since  $\omega(G) \leq \Delta$ , we have  $|S| \geq 2$ . If some two vertices  $u, v \in S$  are not  $\Delta$ -adjacent in  $G - V(K)$ , then  $G_0 = G - V(K) + uv$  has at least  $m$   $\Delta$ -free vertices, and the inequality follows again by Lemma 4 and the induction hypothesis applied to  $G_0$ . Hence, suppose that the vertices of  $S$  are pairwise  $\Delta$ -adjacent. By Lemma 5, we have  $|S| = 2$ . Since the vertices of  $S$  are  $\Delta$ -adjacent in  $G - V(K)$ , they have degree at least  $\Delta - 2$  in  $G - V(K)$ . Since their degree in  $G$  is at most  $\Delta$ , the number of edges between  $V(K)$  and  $S$  is at most 4. Therefore, there are exactly  $4 = \Delta$  edges between  $V(K)$  and  $S$  and both vertices  $u$  and  $v$  of  $S$  have degree exactly  $\Delta - 2$  in  $G - V(K)$ .

Let  $H$  be the 4-tight induced subgraph in  $G - V(K) + uv$  containing the edge  $uv$ . Since both  $u$  and  $v$  have degree at most  $\Delta - 1$  in  $H$ , we conclude that  $H$  is either the clique on 4 vertices or the extended clique with attachments  $u$  and  $v$ . In the former case,  $H' = G[V(K) \cup V(H)]$  is an extended clique with the common neighbors of  $u$  and  $v$  in  $H$  as attachments. In the latter case,  $H' = G[V(K) \cup V(H)]$  is an extended double-clique. In both cases,  $H'$  is a 4-tight induced subgraph of  $G$  distinct from a clique, which is a contradiction.  $\blacktriangleleft$

Next, we prove a similar claim for graphs of maximum degree at most 3. Staton [22] proved that every subcubic triangle-free  $n$ -vertex graph has independence number at least  $5n/14$ . In particular, we have the following.

**► Corollary 8.** *Let  $G$  be an  $n$ -vertex graph of maximum degree at most 3. If  $G$  contains no 3-tight induced subgraph, then  $\alpha(G) \geq \frac{n}{3} + \frac{1}{42}n$ .*

We aim to generalize this result to graphs containing 3-tight subgraphs. In comparison with Lemma 7, we will actually need to strengthen the claim even more, obtaining an increase in the size of the independent set also for some of the vertices contained in triangles.

A *diamond* is an induced subgraph isomorphic to a clique on 4 vertices minus one edge. A *necklace* is either a diamond necklace or a Havel necklace. A diamond in a graph  $G$  is *free*

if it is not contained in an induced subgraph of  $G$  isomorphic to a necklace. For a subcubic graph  $G$ , let  $m(G) = m_1 + m_2$ , where  $m_1$  is the number of 3-free vertices of  $G$  and  $m_2$  is the number of free diamonds of  $G$ . Note that the same bound was obtained in [17] under more restrictive assumptions (only for diamond-free graphs without Havel necklaces and triangle-dominated 6-cycles).

► **Lemma 9.** *If  $G$  is an  $n$ -vertex graph with  $\max(\Delta(G), \omega(G)) \leq 3$ , then  $\alpha(G) \geq \frac{n}{3} + \frac{1}{42}m(G)$ .*

**Proof.** We proceed by induction, and thus we can assume that the claim holds for all graphs with less than  $n$  vertices.

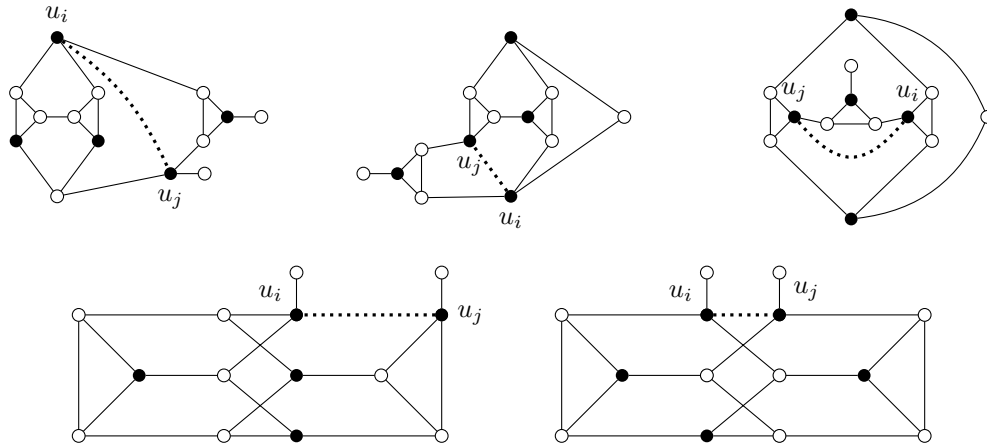
If  $H$  is an induced subgraph of  $G$  isomorphic to a necklace or a triangle-dominated 6-cycle, then observe that  $t = \alpha(H) \in \{2, 3, 4\}$  and  $H$  has an independent set of size  $t$  whose closed neighborhood in  $G$  is contained in  $V(H)$ . Therefore,  $\alpha(G) = \alpha(G - V(H)) + t$ . Furthermore, no vertex of  $V(H)$  is 3-free or contained in a free diamond of  $G$ , and thus  $m(G - V(H)) \geq m(G)$ . Observe that  $|V(H)| = 3t$ , and hence  $\alpha(G) = \alpha(G - V(H)) + t \geq \frac{n-3t}{3} + \frac{1}{42}m(G - V(H)) + t \geq \frac{n}{3} + \frac{1}{42}m(G)$  by the induction hypothesis. Hence, we can without loss of generality assume that the only 3-tight induced subgraphs of  $G$  are triangles.

Suppose that  $H$  is a diamond in  $G$ , which is necessarily free. Let  $G'$  be obtained from  $G$  by contracting all vertices of  $H$  into a single vertex  $v$ . Observe that  $v$  has degree at most 2 in  $G'$ , and thus  $G'$  is  $K_4$ -free. If  $v$  were contained in either a necklace or a triangle in  $G'$ , then  $H$  would be a part of a necklace in  $G$ , contradicting the conclusion of the previous paragraph. Consequently, we replaced a free diamond  $H$  of  $G$  by a 3-free vertex  $v$  in  $G'$ , and thus  $m(G') \geq m(G)$ . Moreover, for every independent set  $S'$  in  $G'$  there exists an independent set  $S$  in  $G$  such that  $|S| = |S'| + 1$  (if  $v \in S'$ , we can add the two non-adjacent vertices of  $H$  to  $S$  instead; if  $v \notin S'$ , then we can add one of the vertices whose degree in  $H$  is 3 to  $S$ ). Hence,  $\alpha(G) \geq \alpha(G') + 1 \geq \frac{n-3}{3} + \frac{1}{42}m(G') + 1 \geq \frac{n}{3} + \frac{1}{42}m(G)$ . Therefore, we can without loss of generality assume that  $G$  is diamond-free.

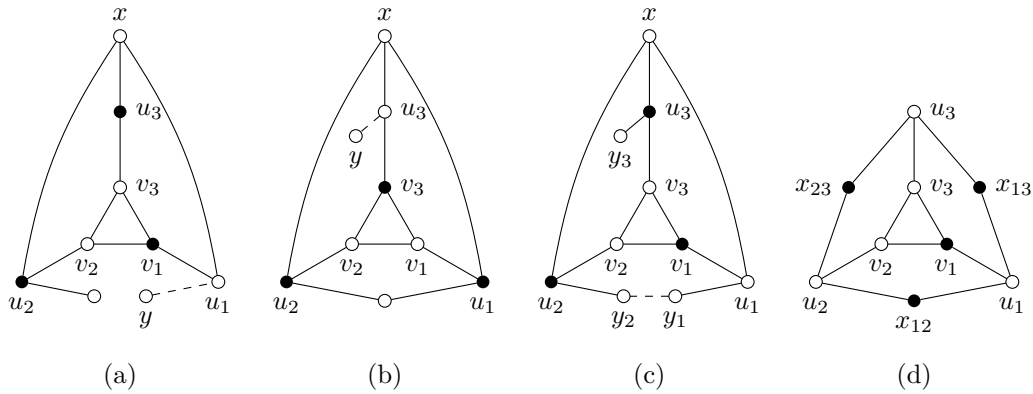
If  $G$  is triangle-free, then  $\alpha(G) \geq \frac{n}{3} + \frac{1}{42}n = \frac{n}{3} + \frac{1}{42}m(G)$  by Corollary 8. Hence, suppose that  $H$  is a triangle in  $G$ , with vertices  $v_1, v_2$ , and  $v_3$ . If say  $v_1$  has degree two in  $G$ , then every independent set in  $G - V(H)$  can be extended to an independent set in  $G$  by including  $v_1$ . Moreover, none of the vertices in  $V(H)$  is 3-free or in a free diamond, which gives  $m(G - V(H)) \geq m(G)$ . Hence,  $\alpha(G) \geq \alpha(G - V(H)) + 1 \geq \frac{n-3}{3} + \frac{1}{42}m(G - V(H)) + 1 \geq \frac{n}{3} + \frac{1}{42}m(G)$ . Therefore, we can without loss of generality assume that  $V(H)$  contains only vertices of degree three.

For  $i \in \{1, 2, 3\}$ , let  $u_i$  denote the neighbor of  $v_i$  not in  $V(H)$ . Since  $G$  is diamond-free,  $u_i$  and  $u_j$  are distinct for all  $1 \leq i < j \leq 3$ . If  $u_i$  is adjacent to  $u_j$  for some  $1 \leq i < j \leq 3$ , then Lemma 4 and the induction hypothesis give  $\alpha(G) \geq \alpha(G - V(H)) + 1 \geq \frac{n-3}{3} + \frac{1}{42}m(G - V(H)) + 1 \geq \frac{n}{3} + \frac{1}{42}m(G)$ . Hence, assume that  $U = \{u_1, u_2, u_3\}$  is an independent set in  $G$ .

Suppose that  $u_i$  and  $u_j$  have no common neighbor for some  $1 \leq i < j \leq 3$ . If  $u_i$  is not 3-adjacent to  $u_j$  in  $G$ , then let  $G'$  be the graph obtained from  $G - V(H)$  by adding the edge  $u_i u_j$ . We have  $m(G) \geq m(G')$ , and Lemma 4 together with the induction hypothesis give  $\alpha(G) \geq \alpha(G') + 1 \geq \frac{n-3}{3} + \frac{1}{42}m(G') + 1 \geq \frac{n}{3} + \frac{1}{42}m(G)$ . Hence, we can assume that  $u_i$  and  $u_j$  are 3-adjacent in  $G$ . Since  $G$  is diamond-free and  $u_i$  and  $u_j$  do not have a common neighbor, we conclude that  $G$  contains an induced subgraph  $H_0$  containing  $u_i$  and  $u_j$  such that  $H_0 + u_i u_j$  is isomorphic to the diamond-free Havel necklace or the triangle-dominated 6-cycle, and  $G$  contains a subgraph  $H_1$  isomorphic to a graph that is either depicted in Figure 2 or obtained from one of the depicted graphs by identifying a vertex of degree 1 with another vertex of degree at most 2. Note that  $|V(H_1)| \leq 14$ , and as shown in the figure,  $H_1$  contains an independent set of size 5 whose closed neighborhood in  $G$  is contained in  $V(H_1)$ . Since at most



■ **Figure 2** Subgraphs arising from a Havel necklace or a triangle-dominated 6-cycle.



■ **Figure 3** Cases in Lemma 9.

6 vertices of  $V(H_1)$  are 3-free in  $G$ , we have  $m(G - V(H_1)) \geq m(G) - 6$ . Together with the induction hypothesis, we obtain  $\alpha(G) \geq \alpha(G - V(H_1)) + 5 \geq \frac{n-14}{3} + \frac{1}{42}m(G - V(H_1)) + 5 \geq \frac{n+1}{3} + \frac{1}{42}(m(G) - 6) > \frac{n}{3} + \frac{1}{42}m(G)$ .

Therefore, we can assume that  $u_i$  and  $u_j$  have a common neighbor for all  $1 \leq i < j \leq 3$ . There are two cases – either these common neighbors are pairwise different, or there exists a common neighbor of all vertices of  $U$ . We first consider the case that there exists a vertex  $x$  adjacent to  $u_1, u_2,$  and  $u_3$ . We distinguish two subcases.

- The first subcase is that either one of vertices of  $U$  has degree two (see Figure 3(a)), or two vertices of  $U$  have a common neighbor distinct from  $x$  (see Figure 3(b)), and consequently  $|N_G[U]| \leq 9$ . If  $|N_G[U]| \leq 8$ , then let  $Z = N_G[U]$ . If  $|N_G[U]| = 9$ , then let  $Z = N_G[U] \setminus \{y\}$ , where  $y$  is a vertex of  $N_G[U] \setminus (V(H) \cup U)$  with only one neighbor in  $U$ . Note that  $|Z| \leq 8$  and at most 5 vertices of  $Z$  are 3-free in  $G$ , and thus  $m(G - Z) \geq m(G) - 5$ . Furthermore, observe that  $G[Z]$  contains an independent set  $U'$  of size 3 such that  $N_G[U'] \subseteq Z$ . Hence,  $\alpha(G) \geq \alpha(G') + 3 \geq \frac{n-8}{3} + \frac{1}{42}(m(G) - 5) + 3 > \frac{n}{3} + \frac{1}{42}m(G)$ .
- The second subcase is that  $|N_G[U]| = 10$ , and thus the vertices of  $U$  have pairwise distinct neighbors not contained in  $V(H) \cup \{x\}$ . For  $1 \leq i \leq 3$ , let  $y_i$  denote the neighbor of  $u_i$  distinct from  $v_i$  and  $x$ , see Figure 3(c). Let  $G'$  be the graph obtained from  $G$  by removing the 8 vertices of  $Y = V(H) \cup U \cup \{x, y_3\}$  and adding the edge  $y_1y_2$  if it is not

already present (since  $G$  is diamond-free, this does not create  $K_4$ ). All vertices of  $G'$  that are 3-free in  $G$  are also 3-free in  $G'$ , unless they belong to a triangle, a necklace, or a triangle-dominated 6-cycle containing the edge  $y_1y_2$ . Note that at most one necklace or triangle-dominated 6-cycle  $Q$  of  $G'$  contains the edge  $y_1y_2$  (only the 6-vertex diamond necklace can intersect another necklace, and this situation cannot arise since  $G$  is diamond-free). Furthermore, all but at most 9 vertices of  $V(Q)$  are contained in a triangle in  $G$ , and at most 5 vertices of  $Y$  are 3-free in  $G$ . Consequently  $m(G') \geq m(G) - 14$ . Consider any independent set  $S$  of  $G'$ . Since  $y_1y_2 \in E(G')$ , we can by symmetry assume that  $y_2 \notin S$ . Hence,  $S \cup \{v_1, u_2, u_3\}$  is an independent set in  $G$ . By the induction hypothesis, this gives  $\alpha(G) \geq \alpha(G') + 3 \geq \frac{n-8}{3} + \frac{1}{42}m(G') + 3 \geq \frac{n+1}{3} + \frac{1}{42}(m(G) - 14) = \frac{n}{3} + \frac{1}{42}m(G)$ .

Secondly, let us consider the case that there is no common neighbor of all vertices of  $U$ . For  $1 \leq i < j \leq 3$ , let  $x_{ij}$  denote a common neighbor of  $u_i$  and  $u_j$ , and observe that  $x_{12}, x_{13}$ , and  $x_{23}$  are three distinct vertices. Since  $G$  does not contain a Havel necklace as an induced subgraph, the set  $\{x_{12}, x_{13}, x_{23}\}$  is independent in  $G$ . Let  $G'$  be obtained from  $G - N_G[V(H)]$  by identifying  $x_{12}, x_{13}$ , and  $x_{23}$  into a new vertex  $x$ . Since  $G$  does not contain a triangle-dominated 6-cycle,  $G'$  is  $K_4$ -free. All vertices of  $G'$  that are 3-free in  $G$  are also 3-free in  $G'$ , unless they belong to a triangle, a necklace, or a triangle-dominated 6-cycle containing  $x$ . Since  $G$  is diamond-free,  $x$  is contained in at most one such subgraph of  $G'$ , and at most 8 vertices of this subgraph other than  $x$  are 3-free in  $G$ . We have  $|V(G)| - |V(G')| = 8$ , but the vertices of  $H$  are not 3-free in  $G$ . Consequently,  $m(G') \geq m(G) - 14$ . Let  $S$  be an independent set in  $G'$ . If  $x \in S$  then  $(S \setminus x) \cup \{x_{12}, x_{13}, x_{23}, v_1\}$  is an independent set of  $G$ , see Figure 3(d). If  $x \notin S$  then  $S \cup \{u_1, u_2, u_3\}$  is an independent set of  $G$ . This gives  $\alpha(G) \geq \alpha(G') + 3 \geq \frac{n-8}{3} + \frac{1}{42}m(G') + 3 \geq \frac{n+1}{3} + \frac{1}{42}(m(G) - 14) = \frac{n}{3} + \frac{1}{42}m(G)$ . ◀

### 3 Proofs of the main results

We need the following corollary of the list-coloring version of Brook's theorem [5, 13].

► **Lemma 10** ([5, 13]). *Let  $L$  be a list assignment for a graph  $G$  such that  $|L(v)| \geq \deg(v)$  for every  $v \in V(G)$ . If  $G$  is not  $L$ -colorable, then  $G$  contains a clique or an odd cycle  $K$  such that all but at most one vertex of  $K$  have lists of size  $\delta(K) = \Delta(K)$ .*

A set  $Z \subseteq V(G)$  is  $\Delta$ -free if either  $\Delta > 3$  and all vertices of  $Z$  are  $\Delta$ -free, or  $\Delta = 3$  and  $Z$  can be partitioned so that each part either induces a free diamond or contains only 3-free vertices. We say that  $G$  is  $K_\Delta$ -partitioned if there exists a partition of the vertices of  $G$  such that each part induces a clique of size  $\Delta$ . We say that a subset  $Z$  of vertices of  $G$  is  $\Delta$ -profitably nibbled if there exists a partition  $Z_1, \dots, Z_r$  of  $Z$  such that for  $a = 1, \dots, r$ , the set  $Z_a$  is  $\Delta$ -profitable in  $G - \bigcup_{i=1}^{a-1} Z_i$  and  $|Z_a| \leq \Delta + 7$ .

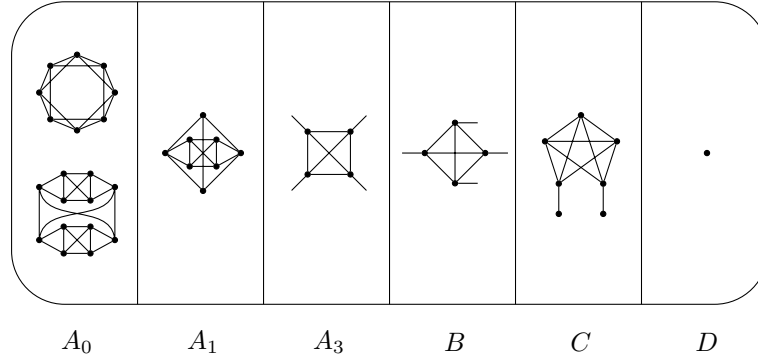
We are now ready to prove our main decomposition result.

► **Lemma 11.** *There exists an algorithm with time complexity  $O(\Delta^2 n)$  that, given as an input an integer  $\Delta \geq 3$  and an  $n$ -vertex graph  $G$  with  $\max(\Delta(G), \omega(G)) \leq \Delta$ , returns a partition of vertices of  $G$  to sets  $A, B, C$ , and  $D$ , such that*

- $G[A]$  is  $\Delta$ -tightly partitioned,
- $G[B]$  is  $K_\Delta$ -partitioned and  $|B| \leq 3\Delta(|C| + |D|)$ ,
- $C$  is  $\Delta$ -profitably nibbled,
- $D$  is  $\Delta$ -free in  $G - C$ , and
- $\alpha(G) = \alpha(G[B \cup C \cup D]) + |A|/\Delta$ .



■ **Figure 4** Set  $Y$  for  $\Delta = 3$  in Lemma 11.



■ **Figure 5** Partition of  $G$  in Lemma 11 for  $\Delta = 4$ .

**Proof.** See Figures 5 and 6 for an illustration of the sets that we construct. Each of the following steps can be easily done in time  $O(\Delta^2 n)$ :

- Initialize  $C = \emptyset$ , and while there exists a set  $Y \subseteq V(G) \setminus C$  such that  $G[Y]$  is isomorphic to
  - if  $\Delta \geq 4$ , the clique on  $\Delta + 1$  vertices minus one edge, or
  - if  $\Delta = 3$ , two diamond necklaces that share an edge or a diamond necklace sharing an edge with a triangle (see Figure 4),
 add  $N_{G-C}[Y]$  to  $C$  (note that  $|N_{G-C}[Y]| \leq \max(\Delta + 3, 10) \leq \Delta + 7$  and  $N_{G-C}[Y]$  is  $\Delta$ -profitable in  $C$ ).
- If  $\Delta \leq 5$ , find all connected components of  $G - C$  of size at most  $4\Delta$  that are  $\Delta$ -tight, and let  $A_0$  be the union of their sets of vertices; otherwise, let  $A_0 = \emptyset$ .
- If  $\Delta = 4$ , find all subgraphs of  $G - (C \cup A_0)$  isomorphic to an extended clique (all these subgraphs are necessarily vertex-disjoint, and vertex-disjoint from all cliques of size 4 not fully contained in the subgraph), and let  $A_1$  be the union of their sets of vertices; otherwise, let  $A_1 = \emptyset$ .
- If  $\Delta = 3$ , find all subgraphs of  $G - (C \cup A_0)$  isomorphic to a necklace (all these subgraphs are necessarily vertex-disjoint and vertex-disjoint from all triangles not fully contained in the subgraph, by the choice of  $C$ ), and let  $A_2$  be the union of their sets of vertices; otherwise, let  $A_2 = \emptyset$ .
- Let  $D$  be the set of vertices of  $G - (A_0 \cup A_1 \cup A_2 \cup C)$  that
  - if  $\Delta \geq 4$  are not contained in cliques of size  $\Delta$ , and
  - if  $\Delta = 3$  are not contained in cliques of size  $\Delta$  or are contained in diamonds.
 Let  $A'_3 = V(G) \setminus (A_0 \cup A_1 \cup A_2 \cup C \cup D)$ .

Note that since  $\Delta(G) \leq \Delta$ , the choice of  $C$  and  $D$  ensures that any two cliques of size  $\Delta$  in  $G[A'_3]$  are vertex-disjoint, and thus  $A'_3$  is  $K_\Delta$ -partitioned. Next, perform the following procedure: initialize  $B = \emptyset$ , and as long as there exists a clique  $K$  of size  $\Delta$  in  $G[A'_3]$  such



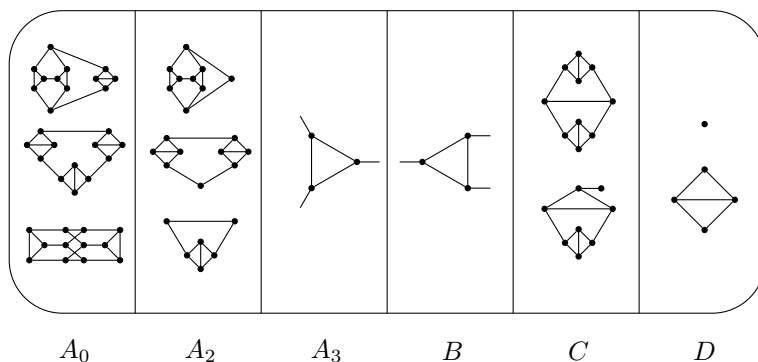


Figure 6 Partition of  $G$  in Lemma 11 for  $\Delta = 3$ .

that at least  $\Delta - 1$  vertices of  $K$  have neighbors in  $B \cup C \cup D$ , remove  $V(K)$  from  $A'_3$  and add it to  $B$ ; let  $A_3$  denote the set obtained from  $A'_3$  by performing this procedure. Note that each step of the procedure decreases the number of edges between  $A'_3$  and  $B \cup C \cup D$  by at least  $\Delta - 2$ , and the number of edges going out of  $|C \cup D|$  is at most  $\Delta(|C| + |D|)$ , and thus  $|B| \leq \frac{\Delta^2}{\Delta-2}(|C| + |D|) \leq 3\Delta(|C| + |D|)$ .

Let  $A = A_0 \cup A_1 \cup A_2 \cup A_3$ . Each  $\Delta$ -tight graph  $H$  satisfies  $\alpha(H) = |V(H)|/\Delta$ , and since  $A$  is  $\Delta$ -tightly partitioned, clearly  $\alpha(G[A]) \leq |A|/\Delta$ . It follows that  $\alpha(G) \leq \alpha(G[B \cup C \cup D]) + |A|/\Delta$ .

Conversely, consider any independent set  $S$  in  $G[B \cup C \cup D]$ . Let  $L$  be the assignment of lists to vertices of  $A$  defined by  $L(v) = \{1, \dots, \Delta\}$  if  $v$  has no neighbor in  $B \cup C \cup D$  and  $L(v) = \{2, \dots, \Delta\}$  otherwise. Note that  $G[A]$  contains no clique  $K$  of size  $\Delta$  such that all but at most one vertex of  $K$  have lists of size  $\Delta - 1$ , as otherwise we would have moved  $K$  to  $B$ . Furthermore, if  $\Delta = 3$ ,  $G[A]$  contains no odd cycle of length at least 5 with all but one vertices having list of size 2, as otherwise  $G[A]$  would not be 3-tightly partitioned. Hence, Lemma 10 implies that  $G[A]$  is  $L$ -colorable. Each  $\Delta$ -tight subgraph  $H$  of  $G[A]$  satisfies  $\alpha(H) \leq |V(H)|/\Delta$ , and since we are using exactly  $\Delta$  colors,  $H$  contains  $|V(H)|/\Delta$  vertices of each color. Since  $A$  is  $\Delta$ -tightly partitioned, the set  $S'$  of vertices of color 1 has size  $|A|/\Delta$ . Hence,  $S \cup S'$  is an independent set in  $G$  of size  $|S| + |A|/\Delta$ . Therefore,  $\alpha(G) = \alpha(G[B \cup C \cup D]) + |A|/\Delta$ . ◀

Next, we show that if  $G$  does not contain independent set much larger than the lower bound  $n/\Delta$ , then the sets  $C$  and  $D$  (and thus also  $B$ ) are small.

► **Lemma 12.** *Let  $\Delta \geq 3$  be an integer and  $k \geq 0$  a rational number, let  $G$  be an  $n$ -vertex graph with  $\max(\Delta(G), \omega(G)) \leq \Delta$ , and let  $A, B, C, D$  be a partition of  $V(G)$  as in Lemma 11. If  $|C| + |D| \geq 34\Delta^2k$ , then  $\alpha(G) \geq n/\Delta + k$  and we can in time  $O(\Delta^2n)$  find an induced subgraph  $G_0$  of  $G$  with  $n_0 \leq 34\Delta^2\lceil k \rceil$  vertices such that  $\alpha(G_0) \geq n_0/\Delta + k$ .*

**Proof.** Let  $C_1, \dots, C_r$  be a partition of  $C$  showing that  $C$  is  $\Delta$ -profitably nibbled. For  $a = 1, \dots, r + 1$  let  $G_a = G - \bigcup_{i=1}^{a-1} C_i$ . For  $1 \leq a \leq r$ , we have  $|C_a| \leq \Delta + 7$  and  $\Delta \cdot (\alpha(G_a) - \alpha(G_{a+1})) \geq |C_a| + 1$ .

Since  $\Delta \geq 3$ , we have  $|C_a| \leq \Delta + 7 \leq 34\Delta$ . Hence  $|C| \leq r34\Delta$ . Moreover,  $\Delta \cdot (\alpha(G) -$

$\alpha(G - C) \geq |C| + r$ , and thus by Lemmas 7 and 9,

$$\begin{aligned} \alpha(G) &\geq \frac{|C| + r}{\Delta} + \alpha(G - C) \geq \frac{|C| + r}{\Delta} + \frac{n - |C|}{\Delta} + \frac{|D|}{34\Delta^2} \\ &= \frac{n}{\Delta} + \frac{r}{\Delta} + \frac{|D|}{34\Delta^2} = \frac{n}{\Delta} + \frac{r34\Delta}{34\Delta^2} + \frac{|D|}{34\Delta^2} \\ &\geq \frac{n}{\Delta} + \frac{|C| + |D|}{34\Delta^2} \geq n/\Delta + k. \end{aligned}$$

The graph  $G_0$  can be defined as  $G_0 = G[C_1 \cup \dots \cup C_{\lceil \Delta k \rceil}]$  if  $r \geq \Delta k$ , and as  $G_0 = G[C \cup D_0]$  for a set  $D_0 \subseteq D$  of size  $\lceil 34\Delta^2 k \rceil - |C|$  otherwise. This can be computed in  $O(\Delta^2 n)$  since the partition of  $V(G)$  to  $A, B, C, D$  can be computed in  $O(\Delta^2 n)$  by Lemma 11. ◀

Finally, we are ready to prove the results stated in the introduction.

**Proof of Theorem 1.** Compute the partition  $A, B, C, D$  of  $V(G)$  as in Lemma 11. By Lemma 12,  $|C| + |D| < 34\Delta^2 k$ , and since  $G[A \cup B]$  is  $\Delta$ -tightly partitioned, we can set  $X = C \cup D$ . ◀

**Proof of Corollary 2.** Compute the partition  $A, B, C, D$  of  $V(G)$  as in Lemma 11, let

$$k = \frac{|C| + |D|}{34\Delta^2},$$

and return that  $\alpha(G) - n/\Delta \geq k$ . That this is a true statement follows from Lemma 12.

On the other hand, since  $G[A \cup B]$  is  $\Delta$ -tightly partitioned,  $\alpha(G) \leq \alpha(G[C \cup D]) + (|A| + |B|)/\Delta \leq |C \cup D| + n/\Delta$ , and thus  $\alpha(G) - n/\Delta \leq |C \cup D| = 34\Delta^2 k$ . Hence, the algorithm approximates  $\alpha(G) - n/\Delta$  up to the factor  $34\Delta^2$ . ◀

**Proof of Corollary 3.** Compute the partition  $A, B, C, D$  of  $V(G)$  as in Lemma 11. If  $|C| + |D| \geq 34\Delta^2 k$ , then return the induced subgraph  $G_0$  obtained in Lemma 12. If  $|C| + |D| < 34\Delta^2 k$ , then  $|B| \leq 3\Delta(|C| + |D|) < 102\Delta^3 k$  by the statement of Lemma 12, and  $|B \cup C \cup D| < 114\Delta^3 k$ . By Lemma 11, the graph  $G_0 = G[B \cup C \cup D]$  satisfies  $\alpha(G_0) - n_0/\Delta = \alpha(G) - n/\Delta$ . ◀

**Acknowledgements.** We would like to thank Andrea Munaro, who found an error in a previous version of this paper.

---

## References

- 1 M. Albertson, B. Bollobás, and S. Tucker. The independence ratio and the maximum degree of a graph. *Congr. Numer.*, 17:43–50, 1976.
- 2 K. Appel and W. Haken. Every planar map is four colorable, Part I: discharging. *Illinois J. of Math.*, 21:429–490, 1977.
- 3 K. Appel, W. Haken, and J. Koch. Every planar map is four colorable, Part II: reducibility. *Illinois J. of Math.*, 21:491–567, 1977.
- 4 Hans L. Bodlaender, Erik Demaine, Michael R. Fellows, Jiong Guo, Danny Hermelin, Daniel Lokshtanov, Moritz Müller, Venkatesh Raman, Johan van Rooij, and Frances A. Rosamond. Open problems in parameterized and exact computation. Technical Report UU-CS-2008-017, Utrecht University, 2008.
- 5 Oleg V. Borodin. Criterion of chromaticity of a degree prescription. In *IV All-Union Conf. on Theoretical Cybernetics (Novosibirsk)*, pages 127–128, 1977.

- 6 Rowland Leonard Brooks. On colouring the nodes of a network. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 37, pages 194–197. Cambridge Univ Press, 1941.
- 7 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*, volume 4. Springer, 2015.
- 8 Rodney G. Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer Science & Business Media, 2012.
- 9 Zdeněk Dvořák and Matthias Mnich. Large independent sets in triangle-free planar graphs. In *Algorithms-ESA 2014*, pages 346–357. Springer, 2014.
- 10 Zdeněk Dvořák and Matthias Mnich. Large independent sets in triangle-free planar graphs. *arXiv e-prints*, 1311.2749v2, 2014.
- 11 Zdenek Dvořák, Jean-Sébastien Sereni, and Jan Volec. Subcubic triangle-free graphs have fractional chromatic number at most  $14/5$ . *Journal of the London Mathematical Society*, 89:641–662, 2014.
- 12 Katherine Edwards and Andrew D. King. Bounding the fractional chromatic number of  $K_\Delta$ -free graphs. *SIAM J. Discrete Math.*, 27(2):1184–1208, 2013.
- 13 P. Erdős, A. L. Rubin, and H. Taylor. Choosability in graphs. *Congr. Numer.*, 26:125–157, 1980.
- 14 Simeon Fajtlowicz. On the size of independent sets in graphs. *Congr. Numer*, 21:269–274, 1978.
- 15 Michael R. Fellows, Jiong Guo, Dániel Marx, and Saket Saurabh. Data Reduction and Problem Kernels (Dagstuhl Seminar 12241). *Dagstuhl Reports*, 2(6):26–50, 2012.
- 16 C. Heckman and R. Thomas. A new proof of the independence ratio of triangle-free cubic graphs. *Discrete Math.*, 233:233–237, 2001.
- 17 Iyad Kanj and Fenghui Zhang. On the independence number of graphs with maximum degree 3. *Theoretical Computer Science*, 478:51–75, 2013.
- 18 Andrew D. King, Linyuan Lu, and Xing Peng. A fractional analogue of Brooks’ theorem. *SIAM Journal on Discrete Mathematics*, 26(2):452–471, 2012.
- 19 Matthias Mnich. Large independent sets in subquartic planar graphs. In *WALCOM: Algorithms and Computation*, pages 209–221. Springer, 2016.
- 20 Rolf Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.
- 21 N. Robertson, P.D. Seymour, and R. Thomas. Quickly excluding a planar graph. *J. Combin. Theory, Ser. B*, 62(2):323–348, 1994.
- 22 W. Staton. Some Ramsey-type numbers and the independence ratio. *Trans. Amer. Math. Soc.*, 256:353–370, 1979.



# Semialgebraic Invariant Synthesis for the Kannan-Lipton Orbit Problem

Nathanaël Fijalkow<sup>1</sup>, Pierre Ohlmann<sup>2</sup>, Joël Ouaknine<sup>3</sup>, Amaury Pouly<sup>4</sup>, and James Worrell<sup>5</sup>

- 1 Department of Computer Science, Oxford University, Oxford, UK
- 2 École Normale Supérieure de Lyon, Lyon, France
- 3 Department of Computer Science, Oxford University, Oxford, UK; and Max Planck Institute for Software Systems (MPI-SWS), Saarland Informatics Campus, Saarbrücken, Germany
- 4 Department of Computer Science, Oxford University, Oxford, UK
- 5 Department of Computer Science, Oxford University, Oxford, UK

---

## Abstract

The *Orbit Problem* consists of determining, given a linear transformation  $A$  on  $\mathbb{Q}^d$ , together with vectors  $x$  and  $y$ , whether the orbit of  $x$  under repeated applications of  $A$  can ever reach  $y$ . This problem was famously shown to be decidable by Kannan and Lipton in the 1980s.

In this paper, we are concerned with the problem of synthesising suitable *invariants*  $\mathcal{P} \subseteq \mathbb{R}^d$ , *i.e.*, sets that are stable under  $A$  and contain  $x$  and not  $y$ , thereby providing compact and versatile certificates of non-reachability. We show that whether a given instance of the Orbit Problem admits a semialgebraic invariant is decidable, and moreover in positive instances we provide an algorithm to synthesise suitable invariants of polynomial size.

It is worth noting that the existence of *semilinear* invariants, on the other hand, is (to the best of our knowledge) not known to be decidable.

**1998 ACM Subject Classification** F.3.1 Specifying and Verifying and Reasoning about Programs

**Keywords and phrases** Verification, algebraic computation, Skolem Problem, Orbit Problem, invariants

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.29

## 1 Introduction

The *Orbit Problem* was introduced by Kannan and Lipton in the seminal papers [8, 9], and shown there to be decidable in polynomial time, answering in the process a decade-old open problem of Harrison on accessibility for linear sequential machines [7]. The Orbit Problem can be stated as follows:

Given a square matrix  $A \in \mathbb{Q}^{d \times d}$  together with vectors  $x, y \in \mathbb{Q}^d$ , decide whether there exists a non-negative integer  $n$  such that  $A^n x = y$ .

In other words, if one considers the discrete ‘orbit’ of the vector  $x$  under repeated applications of the linear transformation  $A$ , does the orbit ever hit the target  $y$ ? Although it is not *a priori* obvious that this problem is even decidable, Kannan and Lipton showed that it can in fact be solved in polynomial time, by making use of spectral techniques as well as some sophisticated results from algebraic number theory.

In instances of non-reachability, a natural and interesting question is whether one can produce a suitable *invariant* as certificate, *i.e.*, a set  $\mathcal{P} \subseteq \mathbb{R}^d$  that is stable under  $A$  (in the



© Nathanaël Fijalkow, Pierre Ohlmann, Joël Ouaknine, Amaury Pouly, and James Worrell; licensed under Creative Commons License CC-BY

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 29; pp. 29:1–29:13

Leibniz International Proceedings in Informatics



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

sense that  $A\mathcal{P} \subseteq \mathcal{P}$ ) and such that  $x \in \mathcal{P}$  and  $y \notin \mathcal{P}$ . The existence of such an invariant then immediately entails by induction that the orbit of  $x$  does indeed avoid  $y$ .

Invariants appear in a wide range of contexts, from gauge theory, dynamical systems, and control theory in physics, mathematics, and engineering to program verification, static analysis, abstract interpretation, and programming language semantics (among others) in computer science. Automated invariant synthesis is a topic of active current research, particularly in the fields of theorem proving and program verification; in the latter, for example, one might imagine that  $y$  corresponds to a faulty or undesirable program state, and an invariant  $\mathcal{P}$  as described above amounts to a succinct ‘safety’ certificate (here the program or procedure in question corresponds to a simple WHILE loop with linear updates).

The widespread use of invariants should not come as a surprise. In addition to their obvious advantage in constituting easily understandable safety certificates, their inductive nature makes them ideally suited to modular reasoning, often allowing one to analyse complex systems by breaking them down into simpler parts, each of which can then be handled in isolation. Invariants, viewed as safety certificates, also enable one to reason over large sets of program states rather than individual instances: in the context of the Orbit Problem, for example, an invariant  $\mathcal{P} \subseteq \mathbb{R}^d$  such that  $x \in \mathcal{P}$  and  $y \notin \mathcal{P}$  doesn’t merely certify that  $y$  is not reachable from  $x$ , but in fact guarantees that from *any* starting point  $x' \in \mathcal{P}$ , it is impossible to reach *any* of the points  $y' \in \mathbb{R}^d \setminus \mathcal{P}$ .

In general, when searching for invariants, one almost always fixes ahead of time a class of suitable potential candidates. Indeed, absent such a restriction, one would point out that the orbit  $\mathcal{O}(x) = \{A^n x : n \geq 0\}$  is always by definition stable under  $A$ , and in instances of non-reachability will therefore always constitute a safety invariant. Such an invariant will however often not be of much use, as it will usually lack good algorithmic properties; for example, as observed in [9], in dimension  $d = 5$  and higher, the question of whether the orbit  $\mathcal{O}(x)$  reaches a given  $(d - 1)$ -dimensional hyperplane corresponds precisely to the famous *Skolem Problem* (of whether an order- $d$  linear recurrence sequence over the integers has a zero), whose decidability has been open for over 80 years [12].

Thus let us assume that we are given a domain  $\mathbf{D} \subseteq 2^{\mathbb{R}^d}$  of suitable potential invariants. At a minimum, one would require that the relevant stability and safety conditions (*i.e.*, for any  $\mathcal{P} \in \mathbf{D}$ , whether  $A\mathcal{P} \subseteq \mathcal{P}$ ,  $x \in \mathcal{P}$ , and  $y \notin \mathcal{P}$ ) be algorithmically checkable (with reasonable complexity). The following natural questions then arise:

1. In instances of non-reachability, does a suitable invariant in  $\mathbf{D}$  *always* exist?
2. If not, can we characterise the exceptional instances in some way?
3. In instances of non-reachability, can we algorithmically determine whether a suitable invariant in  $\mathbf{D}$  exists, and when this is the case can we moreover synthesise such an invariant?

1. and 3. are usually referred to as *completeness* and *relative completeness* respectively, whereas 2. attempts to measure the extent to which completeness fails.

**Main results.** The main results of this paper concern the synthesis of semialgebraic invariants for non-reachability instances of the Kannan-Lipton Orbit Problem, where the input is provided as a triple  $(A, x, y)$  with all entries rational, and can be summarised as follows:

- We prove that whether a suitable semialgebraic<sup>1</sup> invariant exists or not is decidable in

---

<sup>1</sup> A semialgebraic set is the set of solutions of a Boolean combination of polynomial inequalities, with the polynomials in question having integer coefficients.

polynomial space, and moreover in positive instances we show how to synthesise a suitable invariant of polynomial size in polynomial space.

- We provide a simple characterisation of instances of non-reachability for which there does not exist a suitable semialgebraic invariant, and show that such instances are very ‘rare’, in a measure-theoretic sense.

Since the existence of suitable semialgebraic invariants for the Orbit Problem does not coincide precisely with non-reachability, our proof necessarily departs substantially from that given by Kannan and Lipton in [8, 9]. In particular, handling negative instances relies upon certain topological and geometrical insights into the structure of semialgebraic sets, and positive instances require the explicit construction of suitable semialgebraic invariants of polynomial size. We achieve this by making use of techniques from algebraic number theory such as Kronecker’s Theorem on inhomogeneous simultaneous Diophantine approximation, and Masser’s deep results on multiplicative relations among algebraic numbers.

The following three examples illustrate a range of phenomena that arise in searching for semialgebraic invariants.

► **Example 1.** Consider the matrix

$$A = \frac{1}{5} \begin{pmatrix} 4 & -3 \\ 3 & 4 \end{pmatrix}.$$

Matrix  $A$  defines a counterclockwise rotation around the origin by angle  $\arctan(3/5)$ , which is an irrational multiple of  $\pi$ . Thus the topological closure of the orbit  $\mathcal{O} = \{x, Ax, A^2x, \dots\}$  is a circle in  $\mathbb{R}^2$ . If  $y \notin \overline{\mathcal{O}}$  then  $\overline{\mathcal{O}}$  itself is clearly a suitable semialgebraic invariant. On other hand, it can be shown that if  $y \in \overline{\mathcal{O}} \setminus \mathcal{O}$  then there does not exist a suitable semialgebraic invariant. (In passing, it is also not difficult to see that the only polygons  $\mathcal{P}$  that are invariant under  $A$  are  $\emptyset$ ,  $\{(0, 0)\}$ , and  $\mathbb{R}^2$ .) More general orthogonal matrices can be handled along similar lines to the present case, but the analysis is substantially more involved. In general, the only cases in which  $y \notin \mathcal{O}$  but there need not be a semialgebraic invariant are when the matrix  $A$  is diagonalisable and all eigenvalues have modulus one, as in the case at hand.

► **Example 2.** Consider the matrix

$$A = \frac{4}{25} \begin{pmatrix} 4 & -3 & 4 & -3 \\ 3 & 4 & 3 & 4 \\ 0 & 0 & 4 & -3 \\ 0 & 0 & 3 & 4 \end{pmatrix}$$

Matrix  $A$  has spectral radius  $\frac{4}{5}$  and so  $A^n x$  converges to 0 for any initial vector  $x \in \mathbb{Q}^4$ . Given a non-zero target  $y \in \mathbb{Q}^4$  that does not lie in the orbit  $x, Ax, A^2x, \dots$ , a natural candidate for an invariant is an initial segment of the orbit, together with some neighbourhood  $\mathcal{N}$  of the origin in  $\mathbb{R}^4$  that excludes  $y$  and is invariant under  $A$ . Note though that  $A$  is not contractive with respect to either the 1-norm or the 2-norm, so we cannot simply take  $\mathcal{N}$  to be a ball of suitably small radius with respect to either of these norms. However, for  $\varepsilon > 0$ , the set

$$\mathcal{N}_\varepsilon = \{u \in \mathbb{R}^4 : u_1^2 + u_2^2 \leq \varepsilon^2 \wedge u_3^2 + u_4^2 \leq \frac{1}{16}\varepsilon^2\}$$

is invariant under  $A$ . Thus we obtain a semialgebraic invariant as the union of  $\mathcal{N}_\varepsilon$ , where  $\varepsilon$  is chosen sufficiently small such that  $y \notin \mathcal{N}_\varepsilon$ , together with an (easily computable) initial segment of the orbit  $x, Ax, A^2x, \dots$  comprising all points in the orbit that lie outside  $\mathcal{N}_\varepsilon$ .

► **Example 3.** Consider the following scaled version of the matrix from the previous example:

$$A = \frac{1}{5} \begin{pmatrix} 4 & -3 & 4 & -3 \\ 3 & 4 & 3 & 4 \\ 0 & 0 & 4 & -3 \\ 0 & 0 & 3 & 4 \end{pmatrix}.$$

Note that  $A$  is a non-diagonalisable matrix with spectral radius 1. Example 1 concerned an orthogonal matrix, while the matrix in Example 2 was (morally speaking, if not literally) length-decreasing. Here, by contrast, the idea is to identify a subset  $\mathcal{Q} \subseteq \mathbb{R}^4$  that is invariant under  $A$ , together with a “length measure”  $f : \mathcal{Q} \rightarrow \mathbb{R}$  that increases under application of  $A$ . Fixing a constant  $c > 0$ , such a set is

$$\mathcal{Q} = \{u \in \mathbb{R}^4 : u_1^2 + u_2^2 \geq c \wedge u_1 u_3 + u_2 u_4 \geq 0\}$$

with length measure  $f(u) = u_1^2 + u_2^2$ . A key property of  $\mathcal{Q}$  is that for any vector  $x \in \mathbb{R}^4$  such that  $x_3 \neq 0$  or  $x_4 \neq 0$ , the orbit  $x, Ax, A^2x, \dots$  eventually enters  $\mathcal{Q}$ . By choosing  $c$  suitably large, we can exclude  $y$  from  $\mathcal{Q}$ . Thus we obtain an invariant as the union of  $\mathcal{Q}$  and an appropriate finite initial segment of the orbit  $x, Ax, A^2x, \dots$

We would like to draw the reader’s attention to the critical role played by the underlying domain  $\mathbf{D}$  of potential invariants. In the examples above as well as the rest of this paper, we focus exclusively on the domain of semialgebraic sets. However one might naturally consider instead the domain of *semilinear* sets, *i.e.*, sets defined by Boolean combinations of linear inequalities with integer coefficients, or equivalently consisting of finite unions of (bounded or unbounded) rational polytopes. As pointed out above, in Example 1 no non-trivial instance admits a semilinear invariant, whereas one can show that in Example 2 semilinear invariants can always be found. Interestingly, the question of relative completeness (*i.e.*, determining in general whether or not a suitable semilinear invariant exists in non-reachability instances) is not known to be decidable, and appears to be a challenging problem.

## 2 Preliminaries

It is convenient in this paper to work over the field of (complex) algebraic numbers, denoted  $\mathbb{A}$ . All standard algebraic operations, such as sums, products, root-finding of polynomials and computing Jordan normal forms of matrices with algebraic entries can be performed effectively; we refer the reader to [4] for more details on the matter.

An *instance of the Orbit Problem*, or *Orbit instance* for short, is given by a square matrix  $A \in \mathbb{A}^{d \times d}$  and two vectors  $x, y \in \mathbb{A}^d$ . The triple  $(A, x, y)$  is a *reachability* instance if there is  $n \in \mathbb{N}$  such that  $A^n x = y$ , and otherwise is a *non-reachability* instance.

We are interested in non-reachability certificates given as invariants. Formally, given an Orbit instance  $(A, x, y)$  in dimension  $d$ , a set  $\mathcal{P} \subseteq \mathbb{C}^d$  is a *non-reachability invariant* if  $A\mathcal{P} \subseteq \mathcal{P}$ ,  $x \in \mathcal{P}$ , and  $y \notin \mathcal{P}$ .

For the remainder of this paper, we focus on *semialgebraic* invariants. Identifying  $\mathbb{C}^d$  with  $\mathbb{R}^{2d}$ , a set  $\mathcal{P}$  is semialgebraic if it is the set of real solutions of some Boolean combination of polynomial inequalities with integer coefficients.

A central result about semialgebraic sets is the Tarski-Seidenberg Theorem: if  $S \subseteq \mathbb{R}^{n+1}$  is semialgebraic then the image  $\pi(S)$  under the projection  $\pi : \mathbb{R}^{n+1} \rightarrow \mathbb{R}^n$ , where  $\pi(x_1, \dots, x_{n+1}) = (x_1, \dots, x_n)$ , is also semialgebraic. Among the consequences of this result is the fact that the topological closure of a semialgebraic set (in either  $\mathbb{R}^n$  or  $\mathbb{C}^n$ ) is again semialgebraic.



### 3 Semialgebraic Invariants

Our main result is the following.

► **Theorem 4.** *It is decidable whether an Orbit instance admits a semialgebraic invariant. Furthermore, there exists an algorithm which constructs such an invariant when it exists, and the invariant produced has polynomial-size description.*

The remainder of the paper is devoted to proving Theorem 4. To this end, let  $\ell = (A, x, y)$  be a non-reachability Orbit instance in dimension  $d$ .<sup>2</sup>

As a first step, recall that every matrix  $A$  can be written in the form  $A = Q^{-1}JQ$ , where  $Q$  is invertible and  $J$  is in Jordan normal form. The following lemma transfers semialgebraic invariants through the change-of-basis matrix  $Q$ .

► **Lemma 5.** *Let  $\ell = (A, x, y)$  be an Orbit instance, and  $Q$  an invertible matrix in  $\mathbb{A}^{d \times d}$ .*

*Construct the Orbit instance  $\ell_Q = (QAQ^{-1}, Qx, Qy)$ . Then  $\mathcal{P}$  is a semialgebraic invariant for  $\ell_Q$  if, and only if,  $Q^{-1}\mathcal{P}$  is a semialgebraic invariant for  $\ell$ .*

**Proof.** First of all,  $Q^{-1}\mathcal{P}$  is semialgebraic if, and only if,  $\mathcal{P}$  is semialgebraic. We have:

- $QAQ^{-1}\mathcal{P} \subseteq \mathcal{P}$  if, and only if,  $AQ^{-1}\mathcal{P} \subseteq Q^{-1}\mathcal{P}$ ,
- $Qx \in \mathcal{P}$  if, and only if,  $x \in Q^{-1}\mathcal{P}$ ,
- $Qy \notin \mathcal{P}$ , if, and only if,  $y \notin Q^{-1}\mathcal{P}$ .

This concludes the proof. ◀

Thanks to Lemma 5, we can reduce the problem of the existence of semialgebraic invariants for Orbit instances to cases in which the matrix is in Jordan normal form, *i.e.*, is a diagonal block matrix, where the blocks (called Jordan blocks) are of the form:

$$\begin{bmatrix} \lambda & 1 & & \\ & \lambda & \ddots & \\ & & \ddots & 1 \\ & & & \lambda \end{bmatrix}$$

Note that this transformation can be achieved in polynomial time [1, 2].

Formally, a Jordan block is a matrix  $\lambda I + N$  with  $\lambda \in \mathbb{A}$ ,  $I$  the identity matrix and  $N$  the matrix with 1's on the upper diagonal, and 0's everywhere else. The number  $\lambda$  is an eigenvalue of  $A$ . A Jordan block of dimension one is called diagonal, and  $A$  is diagonalisable if, and only if, all Jordan blocks are diagonal.

The  $d$  dimensions of the matrix  $A$  are indexed by pairs  $(J, k)$ , where  $J$  ranges over the Jordan blocks and  $k \in \{1, \dots, \delta\}$  where  $\delta$  is the dimension of the Jordan block  $J$ . For instance, if the matrix  $A$  has two Jordan blocks,  $J_1$  of dimension 1 and  $J_2$  of dimension 2, then the three dimensions of  $A$  are  $(J_1, 1)$  (corresponding to the Jordan block  $J_1$ ) and  $(J_2, 1), (J_2, 2)$  (corresponding to the Jordan block  $J_2$ ).

For a vector  $v$  and a subset  $S$  of  $\{1, \dots, d\}$ , we denote  $v_S$  the projection vector of  $v$  on the dimensions in  $S$ , and extend this notation to matrices. As a special case,  $v_{J, >k}$  denotes the vector restricted to the coordinates of the Jordan block  $J$  whose index is greater than  $k$ . We denote  $\bar{S}$  the complement of  $S$  in  $\{1, \dots, d\}$ .

<sup>2</sup> Kannan and Lipton showed the decidability of reachability for Orbit instances over rational numbers; their proof carries over to instances with algebraic entries, however without the polynomial-time complexity.

There are a few degenerate cases which we handle now. We say that an Orbit instance  $\ell = (A, x, y)$  in Jordan normal form is non-trivial if:

- There is no Jordan block associated with the value 0, or equivalently  $A$  is invertible,
- For each Jordan block  $J$ , both  $x_J$  and  $y_J$  are not the zero vector,
- For each non-diagonal Jordan block  $J$ , the vector  $x_J$  has at least a non-zero coordinate other than the first one, *i.e.*,  $x_{J,>1}$  is not the zero vector.

► **Lemma 6.** *The existence of semialgebraic invariants for Orbit instances reduces in polynomial time to the same problem for non-trivial Orbit instances in Jordan normal form.*

**Proof.** Let  $\ell = (A, x, y)$  be an Orbit instance in Jordan normal form.

- If  $A$  is not invertible, we distinguish two cases.
  - If for some Jordan block  $J$  associated with the eigenvalue 0, we have that  $y$  is not the zero vector, *i.e.*,  $y_J \neq 0$ , then consider  $\mathcal{P} = \{x, Ax, \dots, A^{d-1}x\} \cup \{z \in \mathbb{C}^d \mid z_J = 0\}$  is a semialgebraic invariant. Indeed, the Jordan block  $J$  is nilpotent, so for any vector  $u$  and  $n \geq d$ , we have that  $J^n u = 0$ , so in particular  $(A^n x)_J = 0$ . Moreover, since by assumption  $y$  is not reachable, it is not one of  $A^n x$  for  $n < d$ , and  $y_J \neq 0$ , so  $y \notin \mathcal{P}$ .
  - Otherwise, denote  $J$  the dimensions corresponding to Jordan blocks associated with the eigenvalue 0, we have that  $y_J = 0$ . Consider the Orbit instance  $\ell_J = (A_{\bar{J}}, (A^d x)_{\bar{J}}, y_{\bar{J}})$ . We claim that  $\ell$  admits a semialgebraic invariant if, and only if,  $\ell_J$  does.

Let  $\mathcal{P}$  be a semialgebraic invariant for  $\ell$ . Construct  $\mathcal{P}_J$  the set of vectors  $z$  in  $\mathbb{C}^{\bar{J}}$  such that  $z$  augmented with 0's in the  $J$  dimensions yields a vector in  $\mathcal{P}$ , we argue that  $\mathcal{P}_J$  is a semialgebraic invariant for  $\ell_J$ . Indeed,  $(A^d x)_{\bar{J}} \in \mathcal{P}_J$  since  $A^d x \in \mathcal{P}$  and  $(A^d x)_J = 0$ , because the Jordan block  $J$  is nilpotent. The stability of  $\mathcal{P}_J$  under  $A_{\bar{J}}$  is clear, and  $y_{\bar{J}} \notin \mathcal{P}_J$  because  $y_J = 0$ , so  $y_{\bar{J}} \in \mathcal{P}_J$  would imply  $y \in \mathcal{P}$ .

Conversely, let  $\mathcal{P}_J$  be a semialgebraic invariant for  $\ell_J$ , extend it to  $\mathcal{P} \subseteq \mathbb{C}^d$  by allowing any complex numbers in the  $J$  dimensions, then  $\{x, Ax, \dots, A^{d-1}x\} \cup \mathcal{P}$  is a semialgebraic invariant for  $\ell$ .

We reduced the existence of semialgebraic invariants from  $\ell$  to  $\ell_J$ , with the additional property that the matrix is invertible.

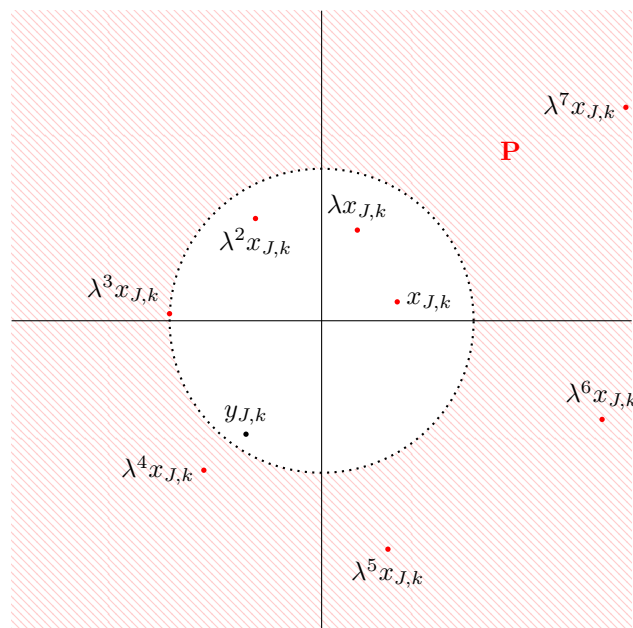
- Suppose  $A$  contains a Jordan block  $J$  such that either  $x_J = 0$  or  $y_J = 0$ . We distinguish three cases.
  - If for some Jordan block  $J$  we have  $x_J = 0$  and  $y_J \neq 0$ , then  $\mathcal{P} = \{z \in \mathbb{C}^d \mid z_J = 0\}$  is a semialgebraic invariant for  $\ell$ .
  - If for some Jordan block  $J$  we have  $x_J \neq 0$  and  $y_J = 0$ , let  $k$  such that  $x_{J,k} \neq 0$  and  $x_{J,>k} = 0$ , then  $\mathcal{P} = \{z \in \mathbb{C}^d \mid z_{J,k} \neq 0 \text{ and } z_{J,>k} = 0\}$  is a semialgebraic invariant for  $\ell$ .
  - Otherwise, denote  $J$  the dimensions corresponding to Jordan blocks for which  $x_J = y_J = 0$ . Consider the Orbit instance  $\ell_J = (A_{\bar{J}}, x_{\bar{J}}, y_{\bar{J}})$ , we claim that  $\ell$  admits a semialgebraic invariant if, and only if,  $\ell_J$  does.

Let  $\mathcal{P}$  be a semialgebraic invariant for  $\ell$ . Construct  $\mathcal{P}_J$  the set of vectors  $z$  in  $\mathbb{C}^{\bar{J}}$  such that  $z$  augmented with 0 in the  $J$  dimensions yields a vector in  $\mathcal{P}$ , then  $\mathcal{P}_J$  is a semialgebraic invariant for  $\ell_J$ .

Conversely, let  $\mathcal{P}_J$  be a semialgebraic invariant for  $\ell_J$ , extend it to  $\mathcal{P} \subseteq \mathbb{C}^d$  by allowing only 0 in the  $J$  dimensions, then  $\mathcal{P}$  is a semialgebraic invariant for  $\ell$ .

We reduced the existence of semialgebraic invariants from  $\ell$  to  $\ell_J$ , with the additional property that for each Jordan block  $J$ , both  $x_J$  and  $y_J$  are not the zero vector.

- If  $A$  contains a non-diagonal Jordan block  $J$  such that the vector  $x_J$  is zero except on the first coordinate  $(J, 1)$ , we distinguish two cases.



■ **Figure 1** Case  $|\lambda| > 1$ . This figure represents the complex plane, which is the projection on the coordinate  $(J, k)$ .

- If for some non-diagonal Jordan block  $J$  we have that  $x_{J,>1} = 0$  and  $y_{J,>1} \neq 0$ , then  $\mathcal{P} = \{z \in \mathbb{C}^d \mid z_{J,>1} = 0\}$  is a semialgebraic invariant for  $\ell$ .
- Otherwise, denote  $J$  the dimensions corresponding to non-diagonal Jordan blocks for which  $x_{J,>1} = y_{J,>1} = 0$ . Let  $S = \bar{J} \cup \bigcup_J (J, 1)$ , i.e., the dimensions outside  $J$  plus the first dimensions of each block in  $J$ . Consider the Orbit instance  $\ell_S = (A_S, x_S, y_S)$ , we claim that  $\ell$  admits a semialgebraic invariant if, and only if,  $\ell_S$  does.

Let  $\mathcal{P}$  be a semialgebraic invariant for  $\ell$ . Construct  $\mathcal{P}_S$  the set of vectors  $z$  in  $\mathbb{C}^S$  such that  $z$  augmented with 0 in the  $\bar{S}$  dimensions yields a vector in  $\mathcal{P}$ , then  $\mathcal{P}_S$  is a semialgebraic invariant for  $\ell_S$ .

Conversely, let  $\mathcal{P}_S$  be a semialgebraic invariant for  $\ell_S$ , extend it to  $\mathcal{P} \subseteq \mathbb{C}^d$  by allowing only 0 in the  $\bar{S}$  dimensions, then  $\mathcal{P}$  is a semialgebraic invariant for  $\ell$ .

We reduced the existence of semialgebraic invariants from  $\ell$  to  $\ell_S$ , with the additional property that for each non-diagonal Jordan block  $J$ ,  $x_{J,>1}$  is not the zero vector.

This concludes the proof. ◀

### 3.1 Some eigenvalue has modulus different from one

► **Lemma 7.** *Let  $\ell = (A, x, y)$  be a non-trivial Orbit instance in Jordan normal form. Assume that  $\ell$  is a non-reachability instance. If the matrix  $A$  has an eigenvalue whose modulus is not equal to 1, then there exists a semialgebraic invariant for  $\ell$ .*

**Proof.** We distinguish two cases according to whether there exists an eigenvalue of modulus strictly more than 1 or an eigenvalue of modulus strictly less than 1.

- Suppose that  $A$  contains a Jordan block  $J$  associated with an eigenvalue  $\lambda$  with  $|\lambda| > 1$ . In this case, some coordinate of  $(A^n x)_{n \in \mathbb{N}}$  diverges to infinity, so eventually gets larger in modulus than the corresponding coordinate in  $y$ . This allows us to construct a

semialgebraic invariant for  $\ell$  by taking the first points and then all points having a large coordinate in the diverging dimension. This case is illustrated in Figure 1.

By assumption  $x_J$  is non-zero, let  $k$  such that  $x_{J,k} \neq 0$  and  $x_{J,>k} = 0$  (Note that if the Jordan block  $J$  is diagonal,  $k = 1$ ). For all  $n \in \mathbb{N}$ , we have  $(A^n x)_{J,k} = \lambda^n x_{J,k}$ , so  $|(A^n x)_{J,k}|$  diverges to infinity. It follows that there exists  $n_0 \in \mathbb{N}$  such that  $|(A^{n_0} x)_{J,k}| > |y_{J,k}|$ . Let

$$\mathcal{P} = \{x, Ax, \dots, A^{n_0-1}x\} \cup \{z \in \mathbb{C}^d \mid |z_{J,k}| \geq |(A^{n_0} x)_{J,k}| \text{ and } z_{J,>k} = 0\}.$$

We argue that  $\mathcal{P}$  is a semialgebraic invariant for  $\ell$ . The non-trivial point is that  $\mathcal{P}$  is stable under  $A$ . Note that  $(A^{n_0} x)_{J,>k} = 0$ , so  $A^{n_0} x \in \mathcal{P}$ . Let  $z \in \mathbb{C}^d$  such that  $|z_{J,k}| \geq |(A^{n_0} x)_{J,k}|$  and  $z_{J,>k} = 0$ . Then  $(Az)_{J,k} = \lambda z_{J,k}$  and  $(Az)_{J,>k} = 0$ , so  $Az \in \mathcal{P}$ .

- If  $A$  contains a Jordan block  $J$  associated with an eigenvalue  $\lambda$  with  $|\lambda| < 1$ .

The situation is similar to the former, except that the convergence is towards the origin. The construction of the semialgebraic invariant is much more subtle though, for the following reason: for  $k$  such that  $x_{J,k} \neq 0$  and  $x_{J,>k} = 0$ , we may have that  $y_{J,k} = 0$ , implying that  $((A^n x)_{J,k})_{n \in \mathbb{N}}$  does not become smaller than  $y_{J,k}$ . Working on another dimension implies to give up the following diagonal behaviour:  $(A^n x)_{J,k} = \lambda^n x_{J,k}$ , making it hard to find a stable set under  $A$ . To overcome this problem, the invariant we define depends upon all the coordinates of the Jordan block  $J$ .

Denote  $d(J)$  the dimension of the Jordan block  $J$ . We have that  $((A^n x)_J)_{n \in \mathbb{N}}$  converges to 0. It follows that there exists  $n_0 \in \mathbb{N}$  such that for each dimension  $k$  of the Jordan block  $J$ , *i.e.*, for  $k \in \{1, \dots, d(J)\}$ , we have  $|(A^{n_0} x)_{J,k}| \leq (1 - |\lambda|)^k \cdot \|y_J\|_\infty$ .

Let

$$\mathcal{P} = \{x, Ax, \dots, A^{n_0-1}x\} \cup \{z \in \mathbb{C}^d \mid \forall k \in \{1, \dots, d(J)\}, |z_{J,k}| \leq (1 - |\lambda|)^k \cdot \|y_J\|_\infty\}.$$

We argue that  $\mathcal{P}$  is a semialgebraic invariant for  $\ell$ . Note that  $y \notin \mathcal{P}$  since for  $k$  such that  $\|y_J\|_\infty = |y_{J,k}|$ , this would imply  $\|y_J\|_\infty \leq (1 - |\lambda|)^k \cdot \|y_J\|_\infty$ , which cannot be since  $k \geq 1$ ,  $y_J \neq 0$  and  $|\lambda| < 1$ . We examine the stability of  $\mathcal{P}$  under  $A$ . Let  $z \in \mathbb{C}^d$  such that for each dimension  $k \in \{1, \dots, d(J)\}$ , we have  $|z_{J,k}| \leq (1 - |\lambda|)^k \cdot \|y_J\|_\infty$ . Let  $k < d(J)$ , then

$$\begin{aligned} |(Az)_{J,k}| = |\lambda z_{J,k} + z_{J,k+1}| &\leq |\lambda| |z_{J,k}| + |z_{J,k+1}| \\ &\leq |\lambda| (1 - |\lambda|)^k \cdot \|y_J\|_\infty + (1 - |\lambda|)^{k+1} \cdot \|y_J\|_\infty \\ &= (|\lambda| + (1 - |\lambda|)) (1 - |\lambda|)^k \cdot \|y_J\|_\infty \\ &= (1 - |\lambda|)^k \cdot \|y_J\|_\infty. \end{aligned}$$

The case  $k = d(J)$  is similar but easier.

This concludes the proof. ◀

### 3.2 All eigenvalues have modulus one and the matrix is not diagonalisable

► **Lemma 8.** *Let  $\ell = (A, x, y)$  be a non-trivial Orbit instance in Jordan normal form and assume that  $\ell$  is a non-reachability instance. If all the eigenvalues of the matrix  $A$  have modulus 1 and  $A$  is not diagonalisable, then there exists a semialgebraic invariant for  $\ell$ .*

We illustrate the construction of the semialgebraic invariant in an example following the proof. (See also Example 3 from the Introduction.)

**Proof.** By assumption, there exists a non-diagonal Jordan block  $J$ . Since  $\ell$  is non-trivial,  $x$  has a non-zero coordinate in  $J$  which is not the first one. Let  $k$  such that  $x_{J,k} \neq 0$  and  $x_{J,>k} = 0$ , we have  $k \geq 2$  and

$$(A^n x)_{J,k-1} = \lambda^n x_{J,k-1} + n\lambda^{n-1} x_{J,k},$$

so  $(|(A^n x)_{J,k-1}|)_{n \in \mathbb{N}}$  diverges to infinity since  $|\lambda| = 1$ . It follows that there exists  $n_0 \in \mathbb{N}$  such that  $|(A^{n_0} x)_{J,k-1}| > |y_{J,k-1}|$ . Without loss of generality we assume  $n_0 \geq -\frac{\langle \lambda x_{J,k-1}, x_{J,k} \rangle}{|x_{J,k}|^2}$ . The notation  $\langle u, v \rangle$  designates the scalar product of the complex numbers  $u$  and  $v$  viewed as vectors in  $\mathbb{R}^2$ , defined by  $\text{Re}(u\bar{v})$ . This quantity will appear later; note that it only depends on  $x$  and  $A$ .

Let

$$\mathcal{P} = \{x, Ax, \dots, A^{n_0-1}x\} \cup \left\{ z \in \mathbb{C}^d \mid \begin{array}{l} |z_{J,k-1}| \geq |(A^{n_0} x)_{J,k-1}|, \text{ and} \\ \langle \lambda z_{J,k-1}, z_{J,k} \rangle \geq 0, \text{ and } z_{J,>k} = 0 \end{array} \right\}.$$

We argue that  $\mathcal{P}$  is a semialgebraic invariant for  $\ell$ . It is a semialgebraic set: the condition  $\langle \lambda z_{J,k-1}, z_{J,k} \rangle \geq 0$  is of the form  $P(z) \geq 0$  for a polynomial  $P$  with algebraic coefficients, where  $z$  is seen as a vector in  $\mathbb{R}^{2d}$ . The part to be looked at closely is the stability of  $\mathcal{P}$  under  $A$ .

First,  $A^{n_0}x \in \mathcal{P}$ . Indeed, using  $|\lambda| = 1$  and the assumption on  $n_0$ :

$$\begin{aligned} \langle \lambda(A^{n_0} x)_{J,k-1}, (A^{n_0} x)_{J,k} \rangle &= \langle \lambda \cdot (\lambda^{n_0} x_{J,k-1} + n_0 \lambda^{n_0-1} x_{J,k}), \lambda^{n_0} x_{J,k} \rangle \\ &= |\lambda^{n_0}|^2 \langle \lambda x_{J,k-1}, x_{J,k} \rangle + n_0 |\lambda^{n_0} x_{J,k}|^2 \\ &= \langle \lambda x_{J,k-1}, x_{J,k} \rangle + n_0 |x_{J,k}|^2 \\ &\geq 0. \end{aligned}$$

Now, let  $z \in \mathbb{C}^d$  such that  $|z_{J,k-1}| \geq |(A^{n_0} x)_{J,k-1}|$ ,  $\langle \lambda z_{J,k-1}, z_{J,k} \rangle \geq 0$  and  $z_{J,>k} = 0$ . We have  $(Az)_{J,k-1} = \lambda z_{J,k-1} + z_{J,k}$ ,  $(Az)_{J,k} = \lambda z_{J,k}$  and  $(Az)_{J,>k} = 0$ . It follows that:

$$\begin{aligned} |(Az)_{J,k-1}|^2 &= |\lambda z_{J,k-1} + z_{J,k}|^2 \\ &= |z_{J,k-1}|^2 + 2\langle \lambda z_{J,k-1}, z_{J,k} \rangle + |z_{J,k}|^2 \\ &\geq |z_{J,k-1}|^2 \\ &\geq |(A^{n_0} x)_{J,k-1}|^2, \end{aligned}$$

and:

$$\begin{aligned} \langle \lambda(Az)_{J,k-1}, (Az)_{J,k} \rangle &= \langle \lambda(\lambda z_{J,k-1} + z_{J,k}), \lambda z_{J,k} \rangle \\ &= |\lambda|^2 \langle \lambda z_{J,k-1} + z_{J,k}, z_{J,k} \rangle \\ &= \langle \lambda z_{J,k-1}, z_{J,k} \rangle + |z_{J,k}|^2 \\ &\geq 0. \end{aligned}$$

Hence  $Az \in \mathcal{P}$ , and  $\mathcal{P}$  is a semialgebraic invariant for  $\ell$ . ◀

► **Example 9.** Consider the following matrix:

$$A = \begin{bmatrix} e^{i\theta} & 1 \\ 0 & e^{i\theta} \end{bmatrix},$$

where  $\theta \in \mathbb{R}$  is an angle such that  $\frac{\theta}{\pi} \notin \mathbb{Q}$ . We start from the vector  $x = [1, 1]^T$ . We have

$$A^n x = \begin{bmatrix} e^{in\theta} + ne^{i(n-1)\theta}, & e^{in\theta} \end{bmatrix},$$

so the projection on the second coordinate is a dense subset of the unit circle, and the projection on the first coordinate describes a growing spiral (similar to that shown in Figure 1). A tentative invariant for excluding some vector  $y$  is the complement of a circle on the first coordinate, large enough not to include  $y$ . However, this set is not a priori invariant. Geometrically, the action of  $A$  on a vector  $[z_1, z_2]$  is to rotate both  $z_1$  and  $z_2$  by an angle of  $\theta$ , and to push the first coordinate in the direction of  $z_2$ :

$$A[z_1, z_2] = [e^{i\theta} z_1 + z_2, e^{i\theta} z_2].$$

A natural way to restrict the above set to make it invariant is to ensure that  $z_2$  pushes away from the origin, *i.e.*, that the norm of  $(Az)_1$  increases. This is achieved by requiring that  $\langle e^{i\theta} z_1, z_2 \rangle \geq 0$ .

### 3.3 All eigenvalues have modulus one and the matrix is diagonalisable

This case is the most involved and is the only one in which it might hold that  $y$  not be reachable and yet no semialgebraic invariant exists. (Recall Example 1 from the Introduction.) Using results from Diophantine approximation and algebraic number theory, we show that the topological closure of the orbit  $\{A^n x : x \in \mathbb{N}\}$  is (effectively) semialgebraic. Furthermore, using topological properties of semialgebraic sets we show that any semialgebraic invariant must contain the closure of the orbit. It follows that there exists a semialgebraic invariant just in case  $y \notin \overline{\{A^n x : x \in \mathbb{N}\}}$ .

We start with the following topological fact about semialgebraic sets.

► **Lemma 10.** *Let  $E, F \subseteq \mathbb{R}^n$  be two sets such that  $\overline{E} = \overline{F}$  and  $F$  is semialgebraic. Then  $E \cap F \neq \emptyset$ .*

**Proof.** The proof uses the notion of the dimension of a semialgebraic set. The formal definition of dimension uses the cell-decomposition theorem (see, e.g., [5, Chapter 4]). However to establish the lemma it suffices to note the following two properties of the dimension. First, for any semi-algebraic set  $X \subseteq \mathbb{R}^n$  set we have  $\dim(X) = \dim(\overline{X})$  [5, Chapter 4, Theorem 1.8]. Secondly, if  $X \subseteq Y$  are semi-algebraic subsets of  $\mathbb{R}^n$  that have the same dimension, then  $X$  has non-empty interior in  $Y$  [5, Chapter 4, Corollary 1.9].

In the situation at hand, since  $\dim(F) = \dim(\overline{F})$  it follows that  $F$  has non-empty interior (with respect to the subspace topology) in  $\overline{F} = \overline{E}$ . But then  $E$  is dense in  $\overline{E}$  while  $F$  has non-empty interior in  $\overline{E}$ , and thus  $E$  and  $F$  meet. ◀

► **Lemma 11.** *Let  $\ell = (A, x, y)$  be an Orbit instance, where  $A = \text{diag}(\lambda_1, \dots, \lambda_d)$  is a diagonal  $d \times d$  matrix with entries  $\lambda_1, \dots, \lambda_d \in \mathbb{C}$  all having modulus one. Write  $\mathcal{O} = \{A^n x : n \in \mathbb{N}\}$  for the orbit of  $x$  under  $A$ . Then*

- *The topological closure of  $\mathcal{O}$  in  $\mathbb{C}^d$  is a semi-algebraic set that is computable from  $\ell$  in polynomial space.*
- *Any semi-algebraic invariant for  $\ell$  contains  $\overline{\mathcal{O}}$ .*

**Proof.** We start by proving the first item.

Write  $\mathbb{T}$  for the unit circle in  $\mathbb{C}$ . Let

$$L_A = \{v \in \mathbb{Z}^d \mid \lambda_1^{v_1} \cdots \lambda_d^{v_d} = 1\}$$

be the set of all multiplicative relations holding among  $\lambda_1, \dots, \lambda_d$ . Notice that  $L_A$  is an additive subgroup of  $\mathbb{Z}^d$ . Consider the set of diagonal  $d \times d$  matrices

$$T_A = \{\text{diag}(\mu_1, \dots, \mu_d) \mid \mu \in \mathbb{T}^d \text{ and } \forall v \in L_A (\mu_1^{v_1} \cdots \mu_d^{v_d} = 1)\}$$

whose diagonal entries satisfy the multiplicative relations in  $L_A$ . Notice that  $T_A$  forms a group under matrix multiplication that is also a closed subset of  $\mathbb{C}^{d \times d}$ .

Using Kronecker's Theorem on inhomogeneous simultaneous Diophantine approximation [3], it is shown in [11, Proposition 3.5] that  $\{A^n : n \in \mathbb{N}\}$  is a dense subset of  $T_A$ . This immediately gives

$$\overline{\mathcal{O}} = \overline{\{A^n x : n \in \mathbb{N}\}} = \{Mx : M \in T_A\}. \tag{1}$$

We now show that  $\overline{\mathcal{O}}$  is semi-algebraic. Observe that  $L_A$  is finitely generated, being a subgroup of a finitely generated group. Moreover, if  $B \subseteq L_A$  is a basis of  $L_A$  then we can write

$$T_A = \{\text{diag}(\mu_1, \dots, \mu_d) \mid \mu \in \mathbb{T}^d \text{ and } \forall v \in B (\mu_1^{v_1} \cdots \mu_d^{v_d} = 1)\}.$$

It follows that  $T_A$  is a semi-algebraic subset of  $\mathbb{C}^{d \times d}$  and thus from (1) that  $\overline{\mathcal{O}}$  is a semi-algebraic set.

From an upper bound on the length of  $B$  due to Masser [10], it can be shown that one can compute a basis for  $L_A$  in polynomial space in the description of  $A$  (see [11, Corollary 3.3]) and thereby compute a description of  $T_A$  as a semi-algebraic set, also in polynomial space in the description of  $A$ .

Now we move to the second item in the statement of the lemma. Let  $\mathcal{P}$  be a semi-algebraic invariant for  $\ell$ . Our goal is to show that  $\overline{\mathcal{O}} \subseteq \mathcal{P}$ . To show this we can, without loss of generality, replace  $\mathcal{P}$  by  $\mathcal{P} \cap \overline{\mathcal{O}}$ , since the latter is also a semi-algebraic invariant. Moreover, since any invariant necessarily contains the orbit  $\mathcal{O}$ , we may suppose that  $\mathcal{O} \subseteq \mathcal{P} \subseteq \overline{\mathcal{O}}$ , and hence  $\overline{\mathcal{P}} = \overline{\mathcal{O}}$ .

We now prove that  $\overline{\mathcal{O}} \subseteq \mathcal{P}$ , that is, we pick an arbitrary element  $z \in \overline{\mathcal{O}}$  and show that  $z \in \mathcal{P}$ . To this end, consider the orbit of  $z$  under the matrix  $A^{-1}$ . Now  $A^{-1} = \text{diag}(\lambda_1^{-1}, \dots, \lambda_d^{-1})$  and we may define groups  $L_{A^{-1}}$  and  $T_{A^{-1}}$  analogously with  $L_A$  and  $T_A$ . In fact it is clear that  $L_A$  and  $L_{A^{-1}}$  coincide (i.e.,  $\lambda_1, \dots, \lambda_d$  satisfy exactly the same multiplicative relations as  $\lambda_1^{-1}, \dots, \lambda_d^{-1}$ ), and hence also  $T_A = T_{A^{-1}}$ .

Now we claim that the following chain of equalities holds:

$$\overline{\{A^{-n}z : n \in \mathbb{N}\}} = \{Mz : M \in T_{A^{-1}}\} \tag{2}$$

$$= \{Mz : M \in T_A\} \tag{3}$$

$$= \{Mx : M \in T_A\} \tag{4}$$

$$= \overline{\mathcal{O}} = \overline{\mathcal{P}}.$$

Indeed, Equation (2) is an instance of (1), but with  $A^{-1}$  and  $z$  in place of  $A$  and  $x$ . Equation (3) follows from the fact that  $T_A = T_{A^{-1}}$ . To see Equation (4), observe from (1) that  $z$  has the form  $M_0x$  for some  $M_0 \in T_A$ . But  $\{MM_0x : M \in T_A\} = \{Mx : M \in T_A\}$  since  $T_A$ , being a group, contains  $M_0^{-1}$ .

Now we have established that

$$\overline{\{A^{-n}z : n \in \mathbb{N}\}} = \overline{\mathcal{P}}.$$

Then by Lemma 10 we have that  $A^{-n}z$  lies in  $\mathcal{P}$  for some  $n \in \mathbb{N}$ . But since  $\mathcal{P}$  is invariant under  $A$  we have  $z \in \mathcal{P}$ . ◀

► **Corollary 12.** *Let the Orbit instance  $\ell$  be as described in Lemma 11. Then  $\ell$  admits a semi-algebraic invariant if and only if  $y \notin \overline{\mathcal{O}}$ .*

**Proof.** If  $y \notin \overline{\mathcal{O}}$ , then  $\overline{\mathcal{O}}$  is a semi-algebraic invariant for  $\ell$  by the first item in Lemma 11. Conversely, if there exists a semi-algebraic invariant  $\mathcal{P}$  for  $\ell$ , then  $\overline{\mathcal{O}} \subseteq \mathcal{P}$  by the second item in Lemma 11, implying that  $y \notin \overline{\mathcal{O}}$ . ◀

### 3.4 Proof of Theorem 4

We now draw together the results of the previous sections to prove our main result, Theorem 4, giving an effective characterisation of the existence of semialgebraic invariants and a procedure to compute such an invariant when it exists.

Let  $\ell = (A, x, y)$  be a non-reachability Orbit instance. First we put  $A$  in Jordan normal form and simplify  $\ell$  to obtain a non-trivial Orbit instance. We then divide into three cases.

- If some eigenvalue of  $A$  has modulus different from 1 then there is a semialgebraic invariant (see Section 3.1).
- If all eigenvalues have modulus 1 and the matrix is not diagonalisable then there is a semialgebraic invariant (see Section 3.2).
- If all eigenvalues have modulus 1 and the matrix is diagonalisable, then there exists a semialgebraic invariant if and only if the topological closure of the orbit  $\{A^n x : n \in \mathbb{N}\}$  is such an invariant, which holds if and only if the closure does not contain  $y$  (see Section 3.3). Note therefore that non-reachability Orbit instances for which there do not exist semialgebraic invariants are extremely sparse.

Thus we obtain an effective characterisation of the class of Orbit instances for which there exists a semialgebraic invariant. Moreover in those cases in which there exists an invariant we have shown how to compute such an invariant in polynomial space.

## 4 Conclusions

This paper is a first step towards the study of invariants for discrete linear dynamical systems. At present, the question of the existence and of the algorithmic synthesis of suitable invariants for higher-dimensional versions of the Orbit Problem (i.e., when the ‘target’  $y$  to be avoided consists of either a vector space, a polytope, or some other higher-dimensional object) is completely open. Given, as pointed out earlier, that reachability questions with high-dimensional targets appear themselves to be very difficult, one does not expect the corresponding invariant synthesis problems to be easy, yet this approach might prove a tractable alternative well worth exploring.

Our main result is a polynomial-space procedure for deciding existence and computing semialgebraic invariants in instances of the Orbit Problem. The only obstacle to obtaining a polynomial-time bound is the problem of computing a basis of the group of all multiplicative relations among a given collection of algebraic numbers  $\alpha_1, \dots, \alpha_d$ , which is not known to be solvable in polynomial time. Less ambitiously one can ask for a polynomial-time procedure to verify a putative relation  $\alpha_1^{n_1} \dots \alpha_d^{n_d} \stackrel{?}{=} 1$ . Assuming that  $\alpha_1, \dots, \alpha_d$  are represented as elements of an explicitly given finite-dimensional algebra  $K$  over  $\mathbb{Q}$ , Ge [6] gave a polynomial-time algorithm for verifying multiplicative relations. In our setting, however, where  $\alpha_1, \dots, \alpha_d$  are roots of the characteristic polynomial of matrix  $A$ , the dimension of  $K$  may be exponential in  $d$ .

---

### References

- 1 Jin-Yi Cai. Computing Jordan normal forms exactly for commuting matrices in polynomial time. Technical report, SUNY at Buffalo, 2000.
- 2 Jin-Yi Cai, Richard J. Lipton, and Yechezkel Zalcstein. The complexity of the A B C problem. *SIAM J. Comput.*, 29(6):1878–1888, 2000. doi:10.1137/S0097539794276853.
- 3 John W.S. Cassels. *An introduction to Diophantine approximation*. Cambridge University Press, 1965.



- 4 Ventsislav Chonev, Joël Ouaknine, and James Worrell. On the complexity of the Orbit Problem. *Journal of the ACM*, 63(3):23, 2016. doi:10.1145/2857050.
- 5 L. P. D. van den Dries. *Tame Topology and O-minimal Structures*. London Mathematical Society Lecture Note Series. Cambridge University Press, May 1998.
- 6 G. Ge. Testing equalities of multiplicative representations in polynomial time. In *Proceedings of SFCS*, pages 422–426. IEEE Computer Society, 1993.
- 7 Michael A. Harrison. *Lectures on linear sequential machines*. New York-Londres, Academic Press, 1969.
- 8 Ravindran Kannan and Richard J. Lipton. The Orbit Problem is decidable. In *Proceedings of STOC*, pages 252–261, 1980. doi:10.1145/800141.804673.
- 9 Ravindran Kannan and Richard J. Lipton. Polynomial-time algorithm for the Orbit Problem. *Journal of the ACM*, 33(4):808–821, 1986. doi:10.1145/6490.6496.
- 10 David W. Masser. Linear relations on algebraic groups. In Alan Baker, editor, *New Advances in Transcendence Theory*, pages 248–262. Cambridge University Press, 1988. doi:10.1017/CB09780511897184.016.
- 11 Joël Ouaknine and James Worrell. Ultimate positivity is decidable for simple linear recurrence sequences. In *Proceedings of ICALP*, pages 330–341, 2014. doi:10.1007/978-3-662-43951-7\_28.
- 12 Terence Tao. *Structure and Randomness*. AMS, 2008.



# The First-Order Logic of Hyperproperties\*

Bernd Finkbeiner<sup>1</sup> and Martin Zimmermann<sup>2</sup>

- 1 Reactive Systems Group, Saarland University, Saarbrücken, Germany  
finkbeiner@react.uni-saarland.de
- 2 Reactive Systems Group, Saarland University, Saarbrücken, Germany  
zimmermann@react.uni-saarland.de

---

## Abstract

We investigate the logical foundations of hyperproperties. Hyperproperties generalize trace properties, which are sets of traces, to sets of sets of traces. The most prominent application of hyperproperties is information flow security: information flow policies characterize the secrecy and integrity of a system by comparing two or more execution traces, for example by comparing the observations made by an external observer on execution traces that result from different values of a secret variable.

In this paper, we establish the first connection between temporal logics for hyperproperties and first-order logic. Kamp’s seminal theorem (in the formulation due to Gabbay et al.) states that linear-time temporal logic (LTL) is expressively equivalent to first-order logic over the natural numbers with order. We introduce first-order logic over sets of traces and prove that HyperLTL, the extension of LTL to hyperproperties, is strictly subsumed by this logic. We furthermore exhibit a fragment that is expressively equivalent to HyperLTL, thereby establishing Kamp’s theorem for hyperproperties.

**1998 ACM Subject Classification** F.4.1 Mathematical Logic

**Keywords and phrases** Hyperproperties, Linear Temporal Logic, First-order Logic

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.30

## 1 Introduction

Linear-time temporal logic (LTL) [19] is one of the most commonly used logics in model checking [2], monitoring [17], and reactive synthesis [10], and a prime example for the “unusual effectiveness of logic in computer science” [16]. LTL pioneered the idea that the correctness of computer programs should not just be specified in terms of a relation between one-time inputs and outputs, but in terms of the infinite sequences of such interactions captured by the *execution traces* of the program. The fundamental properties of the logic, in particular its ultimately periodic model property [21], and the connection to first-order logic via Kamp’s theorem [18], have been studied extensively and are covered in various handbook articles and textbooks (cf. [7, 22]).

In this paper, we revisit these foundations in light of the recent trend to consider not only the individual traces of a computer program, but properties of *sets* of traces, so-called *hyperproperties* [5]. The motivation for the study of hyperproperties comes from information flow security. Information flow policies characterize the secrecy and integrity of a system by relating two or more execution traces, for example by comparing the observations made by

---

\* This work was partially supported by the German Research Foundation (DFG) under the project “SpAGAT” (FI 936/2-3) in the Priority Program 1496 “Reliably Secure Software Systems” and under the project “TriCS” (ZI 1516/1-1).



an external observer on traces that result from different values of a secret variable. Such a comparison can obviously not be expressed as a property of individual traces, but it can be expressed as a property of the full set of system traces. Beyond security, hyperproperties also occur naturally in many other settings, such as the symmetric access to critical resources in distributed protocols, and Hamming distances between code words in coding theory [12].

HyperLTL [4], the extension of LTL to hyperproperties, uses *trace quantifiers* and *trace variables* to refer to multiple traces at the same time. For example, the formula

$$\forall\pi. \forall\pi'. \mathbf{G} (a_\pi \leftrightarrow a_{\pi'}) \quad (1)$$

expresses that *all* computation traces must *agree* on the value of the atomic proposition  $a$  at all times. The extension is useful: it has been shown that most hyperproperties studied in the literature can be expressed in HyperLTL [20]. There has also been some success in extending algorithms for model checking [12], monitoring [1], and satisfiability [11] from LTL to HyperLTL. So far, however, we lack a clear understanding of how deeply the foundations of LTL are affected by the extension. Of particular interest would be a characterization of the models of the logic. Are the models of a satisfiable HyperLTL formula still “simple” in the sense of the ultimately periodic model theorem of LTL?

It turns out that the differences between LTL and HyperLTL are surprisingly profound. Every satisfiable LTL formula has a model that is a (single) ultimately periodic trace. Such models are in particular finite and finitely representable. One might thus conjecture that a satisfiable HyperLTL formula has a model that consists of a finite set of traces, or an  $\omega$ -regular set of traces, or at least *some* set of ultimately periodic traces. In Section 3, we refute *all* these conjectures. Some HyperLTL formulas have only infinite models, some have only non-regular models, and some have only aperiodic models. We can even encode the prime numbers in HyperLTL!

Is there some way, then, to characterize the expressive power of HyperLTL? For LTL, Kamp’s seminal theorem [18] (in the formulation due to Gabbay et al. [14]) states that LTL is expressively equivalent to first-order logic  $FO[<]$  over the natural numbers with order. In order to formulate a corresponding “Kamp’s theorem for HyperLTL,” we have to decide how to encode sets of traces as relational structures, which also induces the signature of the first-order logic we consider. We chose to use relational structures that consist of disjoint copies of the natural numbers with order, one for each trace. To be able to compare positions on different traces, we add the *equal-level predicate*  $E$  (cf. [23]), which relates the same time points on different traces. The HyperLTL formula (1), for example, is equivalent to the  $FO[<, E]$  formula

$$\forall x. \forall y. E(x, y) \rightarrow (P_a(x) \leftrightarrow P_a(y)).$$

In Section 4, we show that  $FO[<, E]$  is *strictly more expressive* than HyperLTL, i.e., every HyperLTL formula can be translated into an equivalent  $FO[<, E]$  formula, but there exist  $FO[<, E]$  formulas that cannot be translated to HyperLTL. Intuitively,  $FO[<, E]$  can express requirements which relate at some point in time an *unbounded* number of traces, which is not possible in HyperLTL. To obtain a fragment of  $FO[<, E]$  that is expressively equivalent to HyperLTL, we must rule out such properties. We consider the fragment where the quantifiers either refer to initial positions or are guarded by a constraint that ensures that the new position is on a trace identified by an initial position chosen earlier. In this way, a formula can only express properties of the bounded number of traces selected by the quantification of initial positions. We call this fragment HyperFO, the *first-order logic of hyperproperties*. Theorem 9, the main result of the paper, then shows that HyperLTL and

HyperFO are indeed expressively equivalent, and thus proves that Kamp's correspondence between temporal logic and first-order logic also holds for hyperproperties.

All proofs omitted due to space restrictions can be found in the full version [13].

## 2 HyperLTL

Fix a finite set AP of atomic propositions. A trace over AP is a map  $t: \mathbb{N} \rightarrow 2^{\text{AP}}$ , denoted by  $t(0)t(1)t(2)\dots$ . The set of all traces over AP is denoted by  $(2^{\text{AP}})^\omega$ . The projection of  $t$  to AP' is the trace  $(t(0) \cap \text{AP}')(t(1) \cap \text{AP}')(t(2) \cap \text{AP}')\dots$  over AP'. A trace  $t$  is ultimately periodic, if  $t = t_0 \cdot t_1^\omega$  for some  $t_0, t_1 \in (2^{\text{AP}})^+$ , i.e., there are  $s, p > 0$  with  $t(n) = t(n+p)$  for all  $n \geq s$ . A set  $T$  of traces is ultimately periodic, if every trace in  $T$  is ultimately periodic.

The formulas of HyperLTL are given by the grammar

$$\begin{aligned} \varphi &::= \exists \pi. \varphi \mid \forall \pi. \varphi \mid \psi \\ \psi &::= a_\pi \mid \neg \psi \mid \psi \vee \psi \mid \mathbf{X} \psi \mid \psi \mathbf{U} \psi \end{aligned}$$

where  $a$  ranges over atomic propositions in AP and where  $\pi$  ranges over a given countable set  $\mathcal{V}$  of *trace variables*. Conjunction, implication, equivalence, and exclusive disjunction  $\oplus$  as well as the temporal operators eventually  $\mathbf{F}$  and always  $\mathbf{G}$  are derived as usual. A sentence is a closed formula, i.e., the formula has no free trace variables.

The semantics of HyperLTL is defined with respect to a trace assignment, a partial mapping  $\Pi: \mathcal{V} \rightarrow (2^{\text{AP}})^\omega$ . The assignment with empty domain is denoted by  $\Pi_\emptyset$ . Given a trace assignment  $\Pi$ , a trace variable  $\pi$ , and a trace  $t$  we denote by  $\Pi[\pi \rightarrow t]$  the assignment that coincides with  $\Pi$  everywhere but at  $\pi$ , which is mapped to  $t$ . Furthermore,  $\Pi[j, \infty]$  denotes the assignment mapping every  $\pi$  in  $\Pi$ 's domain to  $\Pi(\pi)(j)\Pi(\pi)(j+1)\Pi(\pi)(j+2)\dots$ .

For sets  $T$  of traces and trace-assignments  $\Pi$  we define

- $(T, \Pi) \models a_\pi$ , if  $a \in \Pi(\pi)(0)$ ,
- $(T, \Pi) \models \neg \psi$ , if  $(T, \Pi) \not\models \psi$ ,
- $(T, \Pi) \models \psi_1 \vee \psi_2$ , if  $(T, \Pi) \models \psi_1$  or  $(T, \Pi) \models \psi_2$ ,
- $(T, \Pi) \models \mathbf{X} \psi$ , if  $(T, \Pi[1, \infty]) \models \psi$ ,
- $(T, \Pi) \models \psi_1 \mathbf{U} \psi_2$ , if there is a  $j \geq 0$  such that  $(T, \Pi[j, \infty]) \models \psi_2$  and for all  $0 \leq j' < j$ :  $(T, \Pi[j', \infty]) \models \psi_1$ ,
- $(T, \Pi) \models \exists \pi. \varphi$ , if there is a trace  $t \in T$  such that  $(T, \Pi[\pi \rightarrow t]) \models \varphi$ , and
- $(T, \Pi) \models \forall \pi. \varphi$ , if for all traces  $t \in T$ :  $(T, \Pi[\pi \rightarrow t]) \models \varphi$ .

We say that  $T$  satisfies a sentence  $\varphi$ , if  $(T, \Pi_\emptyset) \models \varphi$ . In this case, we write  $T \models \varphi$  and say that  $T$  is a model of  $\varphi$ . Although HyperLTL sentences are required to be in prenex normal form, they are closed under boolean combinations, which can easily be seen by transforming such formulas into prenex normal form.

## 3 The Models of HyperLTL

Every satisfiable LTL formula has an ultimately periodic model, i.e., a particularly simple model: It is trivially finite (and finitely represented) and forms an  $\omega$ -regular language. An obvious question is whether every satisfiable HyperLTL sentence has a simple model, too. Various notions of simplicity could be considered here, e.g., cardinality based ones, being  $\omega$ -regular, or being ultimately periodic, which all extend the notion of simplicity for the LTL case. In this section, we refute all these possibilities: We show that HyperLTL models have to be in general infinite, might necessarily be non-regular, and may necessarily be aperiodic.

### 3.1 No Finite Models

Our first result shows that HyperLTL does not have the finite model property (in the sense that every satisfiable sentence is satisfied by a finite set of traces). The proof is a straightforward encoding of an infinite set of traces that appears again in the following proofs.

► **Theorem 1.** *There is a satisfiable HyperLTL sentence that is not satisfied by any finite set of traces.*

**Proof.** Consider the conjunction  $\varphi$  of the following formulas over  $\text{AP} = \{a\}$ :

- $\forall\pi. (\neg a_\pi) \mathbf{U} (a_\pi \wedge \mathbf{X} \mathbf{G} \neg a_\pi)$ : on every trace there is exactly one occurrence of  $a$ .
- $\exists\pi. a_\pi$ : there is a trace where  $a$  holds true in the first position.
- $\forall\pi. \exists\pi'. \mathbf{F} (a_\pi \wedge \mathbf{X} a_{\pi'})$ : for every trace, say where  $a$  holds at position  $n$  (assuming the first conjunct is satisfied), there is another trace where  $a$  holds at position  $n + 1$ .

It is straightforward to verify that  $\varphi$  is satisfied by the infinite set  $T = \{\emptyset^n \cdot \{a\} \cdot \emptyset^\omega \mid n \geq 0\}$  and an induction over  $n$  shows that every model has to contain  $T$ . Here, one uses the first and second conjunct in the induction start and the first and third conjunct in the induction step. Actually, the first conjunct then implies that  $T$  is the only model of  $\varphi$ . ◀

Next, we complement the lower bound with a matching upper bound.

► **Theorem 2.** *Every satisfiable HyperLTL sentence has a countable model.*

**Proof.** Let  $\varphi$  be a satisfiable HyperLTL sentence and let  $T$  be a model. If  $T$  is countable, then we are done. Thus, assume  $T$  is uncountable and thus in particular non-empty. Furthermore, we assume w.l.o.g.  $\varphi = \forall\pi_0. \exists\pi'_0. \dots \forall\pi_k. \exists\pi'_k. \psi$  with quantifier-free  $\psi$ .

As  $T$  is a model of  $\varphi$ , there is a Skolem function  $f_i: T^i \rightarrow T$  for every  $i \leq k$  satisfying the following property:  $(T, \Pi) \models \psi$  for every trace assignment  $\Pi$  that maps each  $\pi_i$  to some arbitrary  $t_i \in T$  and every  $\pi'_i$  to  $f_i(t_0, \dots, t_i)$ . Note that the relation  $(T, \Pi) \models \psi$  does only depend on  $\Pi$  and  $\psi$ , but not on  $T$ , as  $\psi$  is quantifier-free.

Given a subset  $S \subseteq T$  and a Skolem function  $f_i$  we define

$$f_i(S) = \{f_i(t_0, \dots, t_i) \mid t_0, \dots, t_i \in S\}.$$

Now, fix some  $t \in T$ . Define  $S_0 = \{t\}$  and  $S_{n+1} = S_n \cup \bigcup_{i=0}^k f_i(S_n)$  for every  $n$ , and  $S = \bigcup_{n \geq 0} S_n$ . The limit stage  $S$  is closed under applying the Skolem functions, i.e., if  $t_0, \dots, t_i \in S$ , then  $f_i(t_0, \dots, t_i) \in S$ . Also, every stage  $S_n$  is finite by a straightforward induction, hence  $S$  is countable. We conclude the proof by showing that  $S$  is a model of  $\varphi$ .

Every trace assignment  $\Pi$  mapping  $\pi_i$  to some  $t_i \in S$  and every  $\pi'_i$  to  $f_i(t_0, \dots, t_i) \in S$  satisfies  $(T, \Pi) \models \psi$ , as argued above. Also, as argued above, this is independent of  $T$  due to  $\psi$  being quantifier-free. Hence, we obtain  $(S, \Pi) \models \psi$ . Finally, a simple induction over the quantifier prefix shows  $(S, \Pi_\emptyset) \models \varphi$ , i.e.,  $S$  is indeed a model of  $\varphi$ . ◀

### 3.2 No Regular Models

The construction presented in the proof of Theorem 1, which pushes a single occurrence of the proposition  $a$  through the traces to enforce the set  $\{\emptyset^n \cdot \{a\} \cdot \emptyset^\omega \mid n \geq 0\}$  is reused to prove the main result of this subsection. We combine this construction with an inductive swapping construction to show that HyperLTL sentences do not necessarily have  $\omega$ -regular models. To illustrate the swapping, consider the following finite traces:

$$\begin{array}{ll} t_0 = \{a\} \cdot \emptyset \cdot \{a\} \cdot \emptyset \cdot \{a\} \cdot \emptyset & t_2 = \{a\} \cdot \{a\} \cdot \emptyset \cdot \{a\} \cdot \emptyset \cdot \emptyset \\ t_1 = \{a\} \cdot \{a\} \cdot \emptyset \cdot \emptyset \cdot \{a\} \cdot \emptyset & t_3 = \{a\} \cdot \{a\} \cdot \{a\} \cdot \emptyset \cdot \emptyset \cdot \emptyset \end{array}$$

The trace  $t_1$  is obtained from  $t_0$  by swapping the first occurrence of  $\emptyset$  one position to the right (a swap may only occur between adjacent positions, one where  $a$  holds and one where it does not). Furthermore, with two more swaps, one turns  $t_1$  into  $t_2$  and  $t_2$  into  $t_3$ .

Our following proof is based on the following three observations: (1) In an alternating sequence of even length such as  $t_1$ , the number of positions where  $a$  holds and where  $a$  does not hold is equal. Such a sequence is expressible in (Hyper)LTL. (2) A swap does not change this equality and can be formalized in HyperLTL. (3) Thus, if all occurrences of  $\{a\}$  are swapped to the beginning, then the trace has the form  $\{a\}^n \cdot \emptyset^n$  for some  $n$ . Hence, if we start with all alternating sequences as in  $t_0$ , then we end up with the non-regular language  $\{\{a\}^n \cdot \emptyset^n \mid n > 0\}$ .

► **Theorem 3.** *There is a satisfiable HyperLTL sentence that is not satisfied by any  $\omega$ -regular set of traces.*

**Proof.** Consider the conjunction  $\varphi$  of the formulas  $\varphi_i$ ,  $i \in \{1, \dots, 8\}$  over  $\text{AP} = \{a, b, 1, 2, \dagger\}$ .

$$\blacksquare \varphi_1 = \forall \pi. (1_\pi \oplus 2_\pi) \wedge \neg \dagger_\pi \wedge \neg \dagger_\pi \quad \mathbf{UG} (\dagger_\pi \wedge \neg a_\pi).$$

Every trace from a set of traces satisfying  $\varphi_1$  either satisfies 1 or 2 at the first position. Consequently, we speak of traces of type  $i$  for  $i \in \{1, 2\}$ . Also, on every such trace the truth value of  $\dagger$  changes exactly once, from false to true, after being false at least at the first position. In the following, we are only interested in the unique maximal prefix of a trace where  $\dagger$  does not hold, which we call the *window* of the trace. Note that  $a$  may only hold in the window of a trace. Considering windows essentially turns infinite traces into finite ones.

The balance  $\text{bal}(t)$  of a trace  $t$  is the absolute value of the difference between the number of window positions where  $a$  holds and the number of those where  $a$  does not hold, i.e.,

$$\text{bal}(t) = |\{n \mid a \in t(n) \text{ and } \dagger \notin t(n)\}| - |\{n \mid a \notin t(n) \text{ and } \dagger \notin t(n)\}|.$$

$$\blacksquare \varphi_2 = \forall \pi. 1_\pi \rightarrow (a_\pi \wedge \mathbf{G}(a_\pi \rightarrow \mathbf{X} \neg a_\pi \wedge \mathbf{X} \neg \dagger_\pi \wedge \mathbf{X} \mathbf{X}(a_\pi \vee \dagger_\pi)))$$

$$\blacksquare \varphi_3 = \exists \pi. 1_\pi \wedge a_\pi \wedge \mathbf{X} \mathbf{X} \dagger_\pi$$

$$\blacksquare \varphi_4 = \forall \pi. \exists \pi'. 1_\pi \rightarrow (1_{\pi'} \wedge \mathbf{F}(\neg \dagger_\pi \wedge \mathbf{X} \dagger_\pi \wedge \mathbf{X} \mathbf{X} \neg \dagger_{\pi'} \wedge \mathbf{X} \mathbf{X} \mathbf{X} \dagger_{\pi'}))$$

If  $\varphi_1 \wedge \dots \wedge \varphi_4$  is satisfied by a set of traces, then the projection to  $\{a\}$  of the window of every type 1 trace has the form  $(\{a\} \cdot \emptyset)^n$  for some  $n > 0$ , due to  $\varphi_2$ . In particular, every type 1 trace has balance zero. Furthermore, due to  $\varphi_3$  and  $\varphi_4$ , there is a trace with such a window for every  $n > 0$ .

$$\blacksquare \varphi_5 = \forall \pi. 2_\pi \rightarrow b_\pi \wedge b_\pi \quad \mathbf{UG} \neg b_\pi$$

Finally,  $\varphi_5$  requires every type 2 trace to have a prefix where  $b$  holds true, after which it never holds true again. The length of this prefix is the *rank* of the trace, which is finite.

The next formula implements the swapping process. Each swap has to decrease the rank until a type 1 trace is reached. This rules out models satisfying the formulas by cyclic swaps.

$$\blacksquare \varphi_6 = \forall \pi. \exists \pi'. 2_\pi \rightarrow (\mathbf{F}(\dagger_\pi \wedge \dagger_{\pi'} \wedge \mathbf{X} \neg \dagger_\pi \wedge \mathbf{X} \neg \dagger_{\pi'})) \wedge \varphi_{\text{swp}}(\pi, \pi') \wedge [ \\ (1_{\pi'} \wedge b_\pi \wedge \mathbf{X} \neg b_\pi) \vee \\ (2_{\pi'} \wedge \mathbf{F}(b_{\pi'} \wedge \mathbf{X} \neg b_{\pi'} \wedge \mathbf{X} b_\pi \wedge \mathbf{X} \mathbf{X} \neg b_\pi))]$$

where

$$\varphi_{\text{swp}}(\pi, \pi') = (a_\pi \leftrightarrow a_{\pi'}) \mathbf{U} ((a_\pi \oplus \mathbf{X} a_\pi) \wedge (a_{\pi'} \oplus \mathbf{X} a_{\pi'})) \wedge (a_\pi \oplus a_{\pi'}) \wedge \mathbf{X} \mathbf{X} \mathbf{G}(a_\pi \leftrightarrow a_{\pi'}).$$

Intuitively, this formula requires for every trace  $t$  of type 2 the existence of a trace  $t'$  of the same window length and where the difference in the truth values of  $a$  in  $t$  and  $t'$  is only a single swap at adjacent positions (first line). Furthermore, if  $t$  has rank one, then  $t'$  has to be of type 1 (line two); otherwise, if  $t$  has rank  $r > 1$ , then  $t'$  has to be of type 2 and has to

have rank  $r - 1$  (line three). Thus, the rank is an upper bound on the number of swaps that can be executed before a trace of type 1 is reached.

An induction over the rank of type 2 traces shows that every such trace has balance zero, as a swap as formalized by  $\varphi_{\text{swp}}$  does not change the balance.

- $\varphi_7 = \exists \pi. 2_\pi \wedge a_\pi$
- $\varphi_8 = \forall \pi. \exists \pi'. 2_\pi \rightarrow (2_{\pi'} \wedge (a_\pi \wedge a_{\pi'})) \mathbf{U} (\mathbf{G} \neg a_\pi \wedge a_{\pi'} \wedge \mathbf{X} \mathbf{G} \neg a_{\pi'})$

The last two formulas imply for every  $n > 0$  the existence of a trace of type 2 which has a prefix where  $a$  holds true at exactly the first  $n$  positions, after which it never holds true again. Due to the balance of type 2 traces being zero (assuming all previous formulas are satisfied), the projection to  $\{a\}$  of the window of such a trace has the form  $\{a\}^n \cdot \emptyset^n$ .

Now, towards a contradiction, assume that  $T \models \varphi$  for some  $\omega$ -regular  $T$ . It follows from the observations made above that projecting  $T$  to  $\{a, \dagger\}$  and intersecting it with the  $\omega$ -regular language  $\{a\}^* \cdot \emptyset^* \cdot \{\dagger\}^\omega$  results in the language  $\{\{a\}^n \cdot \emptyset^n \cdot \{\dagger\}^\omega \mid n > 0\}$ , which is not  $\omega$ -regular. This yields the desired contradiction.

To conclude, it suffices to remark that  $\varphi$  is satisfied by taking the union of the set of all required type 1 traces and of the set of all type 2 traces with finite window length, balance zero, and with rank equal to the number of swaps necessary to reach a type 1 trace. ◀

Note that this result can be strengthened by starting with type 1 traces of the form  $(\emptyset \cdot \{a\} \cdot \{a'\} \cdot \{a, a'\})^+ \{\dagger\}^\omega$  for some fresh proposition  $a'$  and then modify the swap operation to obtain sequences of the form  $\emptyset^n \cdot \{a\}^n \cdot \{a'\}^n \cdot \{a, a'\}^n \{\dagger\}^\omega$ . These form, when ranging over all  $n$ , a non- $\omega$ -contextfree language (see [6] for a formal definition of these languages). Thus, not every HyperLTL sentence has an  $\omega$ -contextfree model.

► **Theorem 4.** *There is a satisfiable HyperLTL sentence that is not satisfied by any  $\omega$ -contextfree set of traces.*

It is an interesting question to find a non-trivial class of languages that is rich enough for every satisfiable HyperLTL sentence to be satisfied by a model from this class.

### 3.3 No Periodic Models

Next, we extend the techniques developed in the previous two subsections to show our final result on the complexity of HyperLTL models: although every LTL formula has an ultimately periodic model, one can construct a HyperLTL sentence without ultimately periodic models.

► **Theorem 5.** *There is a satisfiable HyperLTL sentence that is not satisfied by any set of ultimately periodic traces.*

**Proof.** A trace  $t$  is *not* ultimately periodic, if for every  $s, p > 0$  there is an  $n \geq s$  with  $t(n) \neq t(n + p)$ . In the following, we construct auxiliary traces that allow us to express this property in HyperLTL. The main difficulty is to construct traces of the form  $(\{b\}^p \cdot \emptyset^p)^\omega$  for every  $p$ , to implement the quantification of the period length  $p$ .

We construct a sentence  $\varphi$  over  $\text{AP} = \{a, b, 1, 2, \$\}$  with the desired properties, which is a conjunction of several subformulas. The first conjunct requires every trace in a model of  $\varphi$  to have exactly one occurrence of the proposition  $a$ . If it holds at position  $n$ , then we refer to  $n + 1$  as the *characteristic* of the trace (recall that a trace starts at position 0).

As in the proof of Theorem 3, we have two special types of traces in models of  $\varphi$ , which are identified by either 1 or 2 holding true at the first position of every trace, but there might be other traces as well. Type 1 traces are of the form  $\emptyset^c \cdot \{a\} \cdot \emptyset^\omega$  for  $c \geq 0$ . As in the proof



of Theorem 1, one can construct a conjunct that requires the models of  $\varphi$  to contain a type 1 trace for every such  $c$ , but no other traces of type 1.

The projection to  $\{b\}$  of a trace  $t$  of type 2 is a suffix of  $(\{b\}^c \cdot \emptyset^c)^\omega$ , where  $c$  is the characteristic of  $t$ . We claim that one can construct a conjunct of  $\varphi$  that requires all models of  $\varphi$  to contain all these type 2 traces, i.e., all possible suffixes for every  $c > 0$ . This is achieved by formalizing the following properties in HyperLTL:

1. Every type 2 trace has infinitely many positions where  $b$  holds and infinitely many positions where  $b$  does not hold. A block of such a trace is a maximal infix whose positions coincide on their truth values of  $b$ , i.e., either  $b$  holds at every position of the infix, but not at the last one before the infix (if it exists) and not at the first position after the infix or  $b$  does not hold at every position of the infix, but at the last one before it (if it exists) and at the first position after it.
2. For every type 1 trace there is at least one type 2 trace of the same characteristic.
3. The length of the first block of every type 2 trace is not larger than its characteristic.
4. If a block ends at the unique position of a type 2 trace where its  $a$  holds, then it has to be the first block.
5. For every type 2 trace there is another one of the same characteristic that is obtained by shifting the truth values of  $b$  one position to the left.

Assume a set  $T$  of traces satisfies all these properties and assume there is a type 2 trace  $t \in T$  whose projection to  $\{b\}$  is not a suffix of  $(\{b\}^c \cdot \emptyset^c)^\omega$ , where  $c$  is the characteristic of  $t$ . The length of its first block is bounded by  $c$ , due to the third property. Thus, there has to be a non-first block whose length  $\ell$  is not equal to  $c$ . If  $\ell > c$ , we can use the fifth property to shift this block to the left until we obtain a type 2 trace of characteristic  $c$  in  $T$  whose first block has the same length  $\ell$ . This trace violates the third property. If  $\ell < c$ , then we can again shift this block to the left until we obtain a trace in  $T$  of characteristic  $c$  that has a block of length  $\ell$  that ends at the unique position where  $a$  holds. Due to  $\ell < c$ , this cannot be the first block, i.e., we have derived a contradiction to the fourth property.

On the other hand, for every  $c > 0$ , there is a some type 2 trace of characteristic  $c$  in  $T$ . As shown above, its projection to  $\{b\}$  is a suffix of  $(\{b\}^c \cdot \emptyset^c)^\omega$ . Thus, applying the left-shift operation  $2c - 1$  times yields all possible suffixes of  $(\{b\}^c \cdot \emptyset^c)^\omega$ . Thus,  $T$  does indeed contain all possible type 2 traces, if it satisfies the formulas described above.

Recall that we have to express the following property: there is a trace  $t$  such that for every  $s, p > 0$  there is an  $n \geq s$  with  $t(n) \neq t(n + p)$ . To this end, we first existentially quantify a trace  $\pi$  (the supposedly non-ultimately periodic one). Then, we universally quantify two type 1 traces  $\pi_s$  and  $\pi_p$  (thereby fixing  $s$  and  $p$  as the characteristics of  $\pi_s$  and  $\pi_p$ ). Thus, it remains to state that  $\pi$  has two positions  $n$  and  $n'$  satisfying  $s \leq n < n' = n + p$  such that the truth value of  $\$$  differs at these positions. To this end, we need another trace  $\pi'_p$  of the same characteristic  $p$  as  $\pi_p$ , so that a block of  $\pi'_p$  starts at position  $n$ , which allows to determine  $n' = n + p$  by just advancing to the end of the block starting at  $n$ .

Formally, consider the following statement: there is a trace  $\pi$  such that for all type 1 traces  $\pi_s$  and  $\pi_p$  (here, we quantify over  $s$  and  $p$ ) there is a type 2 trace  $\pi'_p$  that has the same characteristic as  $\pi_p$  such that the following is true: there is a position  $n$  no earlier than the one where  $a$  holds in  $\pi_s$  such that

- the truth value of  $b$  in  $\pi'_p$  differs at positions  $n - 1$  and  $n$  (i.e., a block begins at  $n$ ), and
- the atomic proposition  $\$$  holds at  $n$  in  $\pi$  and not at  $n'$  in  $\pi$  or vice versa, where  $n' > n$  is the smallest position such that the truth value of  $b$  in  $\pi'_p$  differs at  $n' - 1$  and  $n'$  (i.e., the next block begins at position  $n'$ ), which implies  $n' = n + p$ .

The formalization of this statement in HyperLTL is the final conjunct of  $\varphi$ . Hence,  $\varphi$  has no models that contain an ultimately periodic trace.

Finally,  $\varphi$  is satisfied by all models that contain all possible type 1 and all possible type 2 traces as well as at least one trace that is not ultimately periodic when projected to  $\{\$$ . ◀

Note that the type 1 and type 2 traces above are ultimately periodic, i.e., although we have formalized the existence of a single non-ultimately periodic trace, the model always has ultimately periodic ones as well. By slightly extending the construction, one can even construct a satisfiable sentence whose models contain not a single ultimately periodic trace. To this end, one requires that every trace (in particular the type 1 and type 2 traces) is non-ultimately periodic, witnessed by the proposition  $\$$  as above.

► **Theorem 6.** *There is a satisfiable HyperLTL sentence that is not satisfied by any set of traces that contains an ultimately periodic trace.*

As a final note on the expressiveness of HyperLTL we show how to encode the prime numbers. Let type 1 and type 2 traces be axiomatized as in the proof of Theorem 5. Recall projecting a type 2 trace to  $\{b\}$  yields a suffix of  $(\{b\}^c \cdot \emptyset^c)^\omega$ , where  $c > 0$  is the trace's characteristic. We say that such a trace is *proper*, if its projection equal to  $(\{b\}^c \cdot \emptyset^c)^\omega$ . Being proper can be expressed in HyperLTL, say by the formula  $\varphi_{\text{prp}}(\pi)$  with a single free variable, relying on the fact that the only occurrence of  $a$  induces the characteristic  $c$ . Also, we add a new atomic proposition  $'$  to AP to encode the prime numbers as follows: the proposition  $'$  holds at the first position of a type 1 trace of characteristic  $c$  if, and only if,  $c$  is a prime number.

Now, consider the following formula, which we add as a new conjunct to the axiomatization of type 1 and type 2 traces:

$$\begin{aligned} & \forall \pi^1. \forall \pi^2. (1_{\pi^1} \wedge '_{\pi^1} \wedge \varphi_{\text{prp}}(\pi^2) \rightarrow \neg \psi(\pi^1, \pi^2)) \wedge \\ & \forall \pi^1. \exists \pi^2. (1_{\pi^1} \wedge \neg '_{\pi^1} \rightarrow \varphi_{\text{prp}}(\pi^2) \wedge \psi(\pi^1, \pi^2)) \end{aligned}$$

Here, the formula  $\psi(\pi^1, \pi^2)$  expresses that the single  $a$  in  $\pi^1$  appears at the end of a non-first block in  $\pi^2$  and that the characteristic of  $\pi^2$  is strictly greater than one. Thus,  $\psi(\pi^1, \pi^2)$  holds if, and only if, the characteristic of  $\pi^2$  is a non-trivial divisor of the characteristic of  $\pi^1$ . Thus, the first conjunct expresses that a type 1 trace of characteristic  $c > 1$  may only have a  $'$  at the first position, if  $c$  has only trivial divisors, i.e., if  $c$  is prime. Similarly, the second conjunct expresses that a type 1 trace of characteristic  $c > 1$  may only not have a  $'$  at the first position, if  $c$  has a non-trivial divisor, i.e., if  $c$  is not prime. Thus, by additionally hardcoding that 1 is not a prime, one obtains a formula  $\varphi$  such that every model  $T$  of  $\varphi$  encodes the primes as follows:  $c$  is prime if, and only if, there is a type 1 trace of characteristic  $c$  in  $T$  with  $'$  holding true at its first position.

## 4 First-order Logic for Hyperproperties

Kamp's seminal theorem [18] states that Linear Temporal Logic with the until-operator  $\mathbf{U}$  and its dual past-time operator “since” is expressively equivalent to first-order logic over the integers with order,  $\text{FO}[\lt]$  for short. Later, Gabbay et al. [14] proved that LTL as introduced here (i.e., exclusively with future-operators) is expressively equivalent to first-order logic over the *natural numbers* with order. More formally, one considers relational structures of the form  $(\mathbb{N}, \lt, (P_a)_{a \in \text{AP}})$  where  $\lt$  is the natural ordering of  $\mathbb{N}$  and each  $P_a$  is a subset of  $\mathbb{N}$ . There is a bijection mapping a trace  $t$  over AP to such a structure  $\underline{t}$ . Furthermore,  $\text{FO}[\lt]$  is

first-order logic<sup>1</sup> over the signature  $\{<\} \cup \{P_a \mid a \in \text{AP}\}$  with equality. The result of Gabbay et al. follows from the existence of the following effective translations: (1) For every LTL formula  $\varphi$  there is an FO[<] sentence  $\varphi'$  such that for all traces  $t$ :  $t \models \varphi$  if, and only if,  $\underline{t} \models \varphi'$ . (2) For every FO[<] sentence  $\varphi$  there is an LTL formula  $\varphi'$  such that for all traces  $t$ :  $\underline{t} \models \varphi$  if, and only if,  $t \models \varphi'$ .

In this section, we investigate whether there is a first-order logic that is expressively equivalent to HyperLTL. The first decision to take is how to represent a set of traces as a relational structure. The natural approach is to take disjoint copies of the natural numbers, one for each trace and label them accordingly. Positions on these traces can be compared using the order. To be able to compare different traces, we additionally introduce a (commutative) equal-level predicate E, which relates the same time points on different traces.

Formally, given a set  $T \subseteq (2^{\text{AP}})^\omega$  of traces over AP, we define the relational structure  $\underline{T} = (T \times \mathbb{N}, <^{\underline{T}}, E^{\underline{T}}, (P_a^{\underline{T}})_{a \in \text{AP}})$  with

- $<^{\underline{T}} = \{((t, n), (t, n')) \mid t \in T \text{ and } n < n' \in \mathbb{N}\}$ ,
- $E^{\underline{T}} = \{((t, n), (t', n)) \mid t, t' \in T \text{ and } n \in \mathbb{N}\}$ , and
- $P_a^{\underline{T}} = \{(t, n) \mid a \in t(n)\}$ .

We consider first-order logic over the signature  $\{<, E\} \cup \{P_a \mid a \in \text{AP}\}$ , i.e., with atomic formulas  $x = y$ ,  $x < y$ ,  $E(x, y)$ , and  $P_a(x)$  for  $a \in \text{AP}$ , and disjunction, conjunction, negation, and existential and universal quantification over elements. We denote this logic by FO[<, E]. We use the shorthand  $x \leq y$  for  $x < y \vee x = y$  and freely use terms like  $x \leq y < z$  with the obvious meaning. A sentence is a closed formula, i.e., every occurrence of a variable is in the scope of a quantifier binding this variable. We write  $\varphi(x_0, \dots, x_n)$  to denote that the free variables of the formula  $\varphi$  are among  $x_0, \dots, x_n$ .

► **Example 7.**

1. The formula  $\text{Succ}(x, y) = x < y \wedge \neg \exists z. x < z < y$  expresses that  $y$  is the direct successor of  $x$  on some trace.
2. The formula  $\text{min}(x) = \neg \exists y. \text{Succ}(y, x)$  expresses that  $x$  is the first position of a trace.

Our first result shows that full FO[<, E] is too expressive to be equivalent to HyperLTL. To this end, we apply a much stronger result due to Bozzelli et al. [3] showing that a certain property expressible in KLTL (LTL with the epistemic knowledge operator **K** [9]) is not expressible in HyperCTL\*, which subsumes HyperLTL.

► **Theorem 8.** *There is an FO[<, E] sentence  $\varphi$  that has no equivalent HyperLTL sentence: For every HyperLTL sentence  $\varphi'$  there are two sets  $T_0$  and  $T_1$  of traces such that*

1.  $\underline{T_0} \not\models \varphi$  and  $\underline{T_1} \models \varphi$ , but
2.  $\varphi'$  cannot distinguish  $T_0$  and  $T_1$ , i.e., either both  $T_0 \models \varphi'$  and  $T_1 \models \varphi'$  or both  $T_0 \not\models \varphi'$  and  $T_1 \not\models \varphi'$ .

**Proof.** Fix  $\text{AP} = \{p\}$  and consider the following property of sets  $T$  of traces over AP: there is an  $n > 0$  such that  $p \notin t(n)$  for every  $t \in T$ . This property is expressible in FO[<, E], but Bozzelli et al. [3] proved that it is not expressible in HyperLTL by constructing sets  $T_0, T_1$  of traces with the desired property.<sup>2</sup> ◀

<sup>1</sup> We assume familiarity with the syntax and semantics of first-order logic. See, e.g., [8], for an introduction to the topic.

<sup>2</sup> Actually, they proved a stronger result showing that the property cannot be expressed in HyperCTL\*, which subsumes HyperLTL. As the latter logic is a branching-time logic, they actually constructed Kripke structures witnessing their result. However, it is easy to show that taking the languages of traces of these Kripke structures proves our claim.

As already noted by Bozzelli et al., the underlying insight is that HyperLTL cannot express requirements which relate at some point in time an unbounded number of traces. By ruling out such properties, we obtain a fragment of  $\text{FO}[\langle, E]$  that is equivalent to HyperLTL. Intuitively, we mimic trace quantification of HyperLTL by quantifying initial positions and then only allow quantification of potentially non-initial positions on the traces already quantified. Thus, such a sentence can only express properties of the bounded number of traces selected by the quantification of initial positions.

To capture this intuition, we have to introduce some notation:  $\exists^M x. \varphi$  is shorthand for  $\exists x. \min(x) \wedge \varphi$  and  $\forall^M x. \varphi$  is shorthand for  $\forall x. \min(x) \rightarrow \varphi$ , i.e., the quantifiers  $\exists^M$  and  $\forall^M$  only range over the first positions of a trace in  $\underline{T}$ . We use these quantifiers to mimic trace quantification in HyperLTL.

Furthermore,  $\exists^G y \geq x. \varphi$  is shorthand for  $\exists y. y \geq x \wedge \varphi$  and  $\forall^G y \geq x. \varphi$  is shorthand for  $\forall y. y \geq x \rightarrow \varphi$ , i.e., the quantifiers  $\exists^G$  and  $\forall^G$  are guarded by a free variable  $x$  and range only over greater-or-equal positions on the same trace that  $x$  is on. We call the free variable  $x$  the *guard* of the quantifier.

We consider sentences of the form

$$\varphi = Q_1^M x_1 \cdots Q_k^M x_k. Q_1^G y_1 \geq x_{g_1} \cdots Q_\ell^G y_\ell \geq x_{g_\ell}. \psi \quad (2)$$

with  $Q \in \{\exists, \forall\}$ , where we require the sets  $\{x_1, \dots, x_k\}$  and  $\{y_1, \dots, y_\ell\}$  to be disjoint, every guard  $x_{g_j}$  to be in  $\{x_1, \dots, x_k\}$ , and  $\psi$  to be quantifier-free with free variables among the  $\{y_1, \dots, y_\ell\}$ . We call this fragment HyperFO. Note that the subformula starting with the quantifier  $Q_1^G$  being in prenex normal form and  $\psi$  only containing the variables  $y_j$  simplifies our reasoning later on, but is not a restriction.

► **Theorem 9.** *HyperLTL and HyperFO are equally expressive.*

We prove this result by presenting effective translations between HyperLTL and HyperFO (see Lemma 12 and Lemma 13). We begin with the direction from HyperFO to HyperLTL. Consider a HyperFO sentence  $\varphi$  as in (2). It quantifies  $k$  traces with the quantifiers  $\exists^M$  and  $\forall^M$ . Every other quantification is then on one of these traces. As trace quantification is possible in HyperLTL, we only have to take care of the subformula starting with the guarded quantifiers. After replacing these quantifiers by unguarded ones, we only have to remove the equal-level predicate to obtain an  $\text{FO}[\langle]$  sentence. To this end, we merge the  $k$  traces under consideration into a single one, which reduces the equal-level predicate to the equality predicate (cf. [23]). The resulting sentence is then translated into LTL using the theorem of Gabbay et al., the merging is undone, and the quantifier prefix is added again. We show that the resulting sentence is equivalent to the original one.

Fix a HyperFO sentence  $\varphi$  as in (2) and consider the subformula

$$\chi = Q_1^G y_1 \geq x_{g_1} \cdots Q_\ell^G y_\ell \geq x_{g_\ell}. \psi$$

obtained by removing the quantification of the guards. We execute the following replacements to obtain the formula  $\chi_m$ :

1. Replace every guarded existential quantification  $\exists^G y_j \geq x_{g_j}$  by  $\exists y_j$  and every guarded universal quantification  $\forall^G y_j \geq x_{g_j}$  by  $\forall y_j$ .
2. Replace every atomic formula  $P_a(y_j)$  by  $P_{(a, g_j)}(y_j)$ , where  $x_{g_j}$  is the guard of  $y_j$ .
3. Replace every atomic formula  $E(y_j, y_{j'})$  by  $y_j = y_{j'}$ .

As we have removed all occurrences of the free guards, the resulting formula  $\chi_m$  is actually a sentence over the signature  $\{\langle\} \cup \{P_a \mid a \in \text{AP} \times \{1, \dots, k\}\}$ , i.e., an  $\text{FO}[\langle]$  sentence.

Given a list  $(t_1, \dots, t_k)$  of traces over AP, define the trace  $\text{mrg}(t_1, \dots, t_k) = A_0 A_1 A_2 \cdots$  over  $\text{AP} \times \{1, \dots, k\}$  via  $A_n = \bigcup_{j=1}^k t_j(n) \times \{j\}$ , i.e., we merge the  $t_j$  into a single trace.

► **Claim 10.** *Let  $T$  be a set of traces and let  $\beta_0: \{x_1, \dots, x_k\} \rightarrow T \times \{0\}$  be a variable valuation of the guards  $x_1, \dots, x_k$  to elements of  $\underline{T}$ . Then,  $(\underline{T}, \beta_0) \models \chi$  if, and only if,  $\text{mrg}(t_1, \dots, t_k) \models \chi_m$ , where  $t_j$  is the unique trace satisfying  $\beta_0(x_{g_j}) = (t_j, 0)$ .*

This claim can be proven by translating a winning strategy for either player in the model checking game [15] for  $(\underline{T}, \chi)$  (starting with the initial variable valuation  $\beta_0$ ) into a winning strategy for the same player in the model checking game for  $(\text{mrg}(t_1, \dots, t_k), \chi_m)$ .

Now, we apply the theorem of Gabbay et al. [14] to  $\chi_m$  and obtain an LTL formula  $\chi'_m$  over  $\text{AP} \times \{1, \dots, k\}$  that is equivalent to  $\chi_m$ . Let  $\chi'$  be the HyperLTL formula obtained from  $\chi'_m$  by replacing every atomic proposition  $(a, j)$  by  $a_{\pi_j}$ , i.e., we undo the merging. The following claim is proven by a simple structural induction over  $\chi_m$ .

► **Claim 11.** *Let  $T$  be a set of traces and let  $\Pi: \{\pi_1, \dots, \pi_k\} \rightarrow T$  be a trace assignment. Then,  $\text{mrg}(\Pi(\pi_1), \dots, \Pi(\pi_k)) \models \chi'_m$  if, and only if,  $(T, \Pi) \models \chi'$ .*

Now, we add the quantifier prefix  $Q_1\pi_1 \dots Q_k\pi_k$ . to  $\chi'$ , where  $Q_j = \exists$ , if  $Q_j^M = \exists^M$ , and  $Q_j = \forall$ , if  $Q_j^M = \forall^M$ . Call the obtained HyperLTL sentence  $\varphi'$ .

► **Lemma 12.** *For every HyperFO sentence  $\varphi$ , there is a HyperLTL sentence  $\varphi'$  such that for every  $T \subseteq (2^{\text{AP}})^\omega$ :  $\underline{T} \models \varphi$  if, and only if,  $T \models \varphi'$ .*

**Proof.** Fix a HyperFO sentence  $\varphi$  and let the  $\chi$ ,  $\chi_m$ ,  $\chi'_m$ ,  $\chi'$ , and  $\varphi'$  be as constructed as above. Let  $\beta_0$  be a variable valuation as in Claim 10, let the traces  $t_1, \dots, t_k \in T$  be defined as in this claim, and let the trace assignment  $\Pi$  map  $\pi_j$  to  $t_j$ .

Then, the following equivalences hold:

$$(\underline{T}, \beta_0) \models \chi \stackrel{\text{Claim 10}}{\Leftrightarrow} \text{mrg}(t_1, \dots, t_k) \models \chi_m \stackrel{\text{by def.}}{\Leftrightarrow} \text{mrg}(t_1, \dots, t_k) \models \chi'_m \stackrel{\text{Claim 11}}{\Leftrightarrow} (T, \Pi) \models \chi'.$$

Finally, the equivalence of  $\varphi$  and  $\varphi'$  follows from the fact that one can identify quantification of initial elements of paths in  $\underline{T}$  and trace quantification in  $T$ , as both  $\varphi$  and  $\varphi'$  have the *same* quantifier prefix. ◀

It remains to consider the translation of HyperLTL into HyperFO, which is straightforward, as usual.

► **Lemma 13.** *For every HyperLTL sentence  $\varphi$ , there is a HyperFO sentence  $\varphi'$  such that for every  $T \subseteq (2^{\text{AP}})^\omega$ :  $T \models \varphi$  if, and only if,  $\underline{T} \models \varphi'$ .*

**Proof.** Let  $\pi_1, \dots, \pi_k$  be the trace variables appearing in  $\varphi$  and fix a set  $G = \{x_1, \dots, x_k, x_t\}$  of first-order variables, which we use as guards: the  $x_j$  with  $j \leq k$  are identified with the trace variables and we use variables guarded by  $x_t$  to model the flow of time. We inductively construct a formula  $\text{fo}(\varphi)$  satisfying the following invariant: For each subformula  $\psi$  of  $\varphi$ , the free variables of the formula  $\text{fo}(\psi)$  comprise of a subset of  $G$  and one additional (different!) variable, which we call the time-variable of  $\text{fo}(\psi)$ . We require the time-variables of the subformulas to be fresh unless stated otherwise and also different from the guards in  $G$ . Intuitively, the time-variables are used to mimic the flow of time when translating a temporal operator. Formally, we define:

- $\text{fo}(a_{\pi_j}) = \exists^G y \geq x_j. E(y, z) \wedge P_a(y)$ , i.e.,  $z$  is the time-variable of  $\text{fo}(a_{\pi_j})$ .
- $\text{fo}(\neg\psi_1) = \neg\text{fo}(\psi_1)$ , i.e., the time-variable is unchanged.
- $\text{fo}(\psi_1 \vee \psi_2) = \text{fo}(\psi'_1) \vee \text{fo}(\psi_2)$ , where we assume w.l.o.g. that  $\text{fo}(\psi_1)$  and  $\text{fo}(\psi'_2)$  have the same time-variable, which is also the time-variable of the disjunction.
- $\text{fo}(\mathbf{X}\psi_1) = \exists^G z_1 \geq x_t. \text{Succ}(z, z_1) \wedge \text{fo}(\psi_1)$ , where  $z_1$  is the time-variable of  $\text{fo}(\psi_1)$ . Hence,  $z$  is the time-variable of  $\text{fo}(\mathbf{X}\psi_1)$ .

- $\text{fo}(\psi_1 \mathbf{U} \psi_2) = \exists^G z_2 \geq x_t. z \leq z_2 \wedge \text{fo}(\psi_2) \wedge \forall^G z_1 \geq x_t. z \leq z_1 < z_2 \rightarrow \text{fo}(\psi_1)$ , where  $z_i$  is the time-variable of  $\text{fo}(\psi_i)$ . Hence,  $z$  is the time-variable of  $\text{fo}(\psi_1 \mathbf{U} \psi_2)$ .
- $\text{fo}(\exists \pi_j. \psi) = \exists^M x_j. \text{fo}(\psi)$ , i.e., the time-variable is unchanged.
- $\text{fo}(\forall \pi_j. \psi) = \forall^M x_j. \text{fo}(\psi)$ , i.e., the time-variable is unchanged.

Now, we define  $\varphi' = \exists^M x_t. \exists^M z. x_t = z \wedge \text{fo}(\varphi)$ , where  $z$  is the time-variable of  $\text{fo}(\varphi)$ . It is straightforward to show that  $\varphi'$  is equivalent to  $\varphi$ . Finally,  $\varphi'$  can be rewritten into prenex normal form (with quantifiers  $Q^M$  and  $Q^G!$ ) so that the outermost quantifiers bind the guards while the inner ones are guarded. ◀

## 5 Conclusion and Discussion

The extension from LTL to HyperLTL has fundamentally changed the models of the logic. While a satisfiable LTL formula is guaranteed to have an ultimately periodic model, we have shown that there is no guarantee that a satisfiable HyperLTL formula has a model that is finite,  $\omega$ -regular, or even just  $\omega$ -contextfree. Characterizing the expressive power of HyperLTL is thus a formidable challenge. Nevertheless, the results of this paper provide a first such characterization. With the definition of  $FO[<, E]$  and HyperFO, and the resulting formulation and proof of Kamp's theorem for hyperproperties, we have established the first connection between temporal logics for hyperproperties and first-order logic. This connection provides a strong basis for a systematic exploration of the models of hyperproperties.

While hyperproperties have recently received a lot of attention from a practical perspective (cf. [1, 4, 12]), their logical and language-theoretic foundations are far less understood, and it is our hope that this paper will attract more research into this exciting area. An important open problem is to find a non-trivial class of languages so that every satisfiable HyperLTL formula is guaranteed to be satisfied by a model from this class. In Section 3, we have ruled out some of the obvious candidates for such a class of languages, such as the  $\omega$ -regular and  $\omega$ -contextfree languages. The challenge remains to identify a class of languages that is rich enough for every satisfiable HyperLTL formula.

Another major open problem is to find a temporal logic that is expressively equivalent to  $FO[<, E]$ . In Section 4, we have shown that HyperLTL is less expressive than  $FO[<, E]$ , by arguing that HyperLTL cannot express requirements which relate at some point in time an unbounded number of traces. Since KLTL [9] can express such properties, KLTL and related epistemic temporal logics are natural candidates for logics that are expressively equivalent to  $FO[<, E]$ . Another promising candidate is HyperLTL with past operators, motivated by the results on HyperCTL\* with past [3].

**Acknowledgements.** We thank Markus N. Rabe and Leander Tentrup for fruitful discussions.

---

## References

- 1 Shreya Agrawal and Borzoo Bonakdarpour. Runtime verification of  $k$ -safety hyperproperties in HyperLTL. In *CSF 2016*, pages 239–252. IEEE Computer Society, 2016. doi:10.1109/CSF.2016.24.
- 2 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.
- 3 Laura Bozzelli, Bastien Maubert, and Sophie Pinchinat. Unifying hyper and epistemic temporal logics. In A. M. Pitts, editor, *FoSSaCS 2015*, volume 9034 of *Lecture Notes in Computer Science*, pages 167–182. Springer, 2015. doi:10.1007/978-3-662-46678-0\_11.

- 4 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In Martín Abadi and Steve Kremer, editors, *POST 2014*, volume 8414 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2014. doi:10.1007/978-3-642-54792-8\_15.
- 5 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.
- 6 Rina S. Cohen and Arie Y. Gold. Theory of omega-languages. I. characterizations of omega-context-free languages. *Journal of Computer and System Sciences*, 15(2):169–184, 1977. doi:10.1016/S0022-0000(77)80004-4.
- 7 Stéphane Demri, Valentin Goranko, and Martin Lange. *Temporal Logics in Computer Science*. Cambridge University Press, 2016.
- 8 Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Mathematical Logic (2. ed.)*. Undergraduate texts in mathematics. Springer, 1994.
- 9 Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.
- 10 Bernd Finkbeiner. Synthesis of reactive systems. In Javier Esparza, Orna Grumberg, and Salomon Sickert, editors, *Dependable Software Systems Engineering*, volume 45 of *NATO Science for Peace and Security Series – D: Information and Communication Security*, pages 72–98. IOS Press, 2016. doi:10.3233/978-1-61499-627-9-72.
- 11 Bernd Finkbeiner and Christopher Hahn. Deciding hyperproperties. In Josée Desharnais and Radha Jagadeesan, editors, *CONCUR 2016*, volume 59 of *LIPICs*, pages 13:1–13:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.CONCUR.2016.13.
- 12 Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking HyperLTL and HyperCTL\*. In Daniel Kroening and Corina S. Pasareanu, editors, *CAV 2015 (Part I)*, volume 9206 of *Lecture Notes in Computer Science*, pages 30–48. Springer, 2015. doi:10.1007/978-3-319-21690-4\_3.
- 13 Bernd Finkbeiner and Martin Zimmermann. The first-order logic of hyperproperties. *arXiv*, 1610.04388, 2016. URL: <http://arxiv.org/abs/1610.04388>.
- 14 Dov M. Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal basis of fairness. In Paul W. Abrahams, Richard J. Lipton, and Stephen R. Bourne, editors, *POPL 1980*, pages 163–173. ACM Press, 1980. doi:10.1145/567446.567462.
- 15 Erich Grädel, Phokion G. Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema, and Scott Weinstein. *Finite Model Theory and Its Applications*. Springer, 2005.
- 16 Joseph Y. Halpern, Robert Harper, Neil Immerman, Phokion G. Kolaitis, Moshe Y. Vardi, and Victor Vianu. On the unusual effectiveness of logic in computer science. *Bulletin of Symbolic Logic*, 7(2):213–236, 2001. doi:10.2307/2687775.
- 17 Klaus Havelund and Grigore Rosu. Efficient monitoring of safety properties. *International Journal on Software Tools for Technology Transfer*, 6(2):158–173, 2004. doi:10.1007/s10009-003-0117-6.
- 18 Hans W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, Computer Science Department, University of California at Los Angeles, USA, 1968.
- 19 Amir Pnueli. The Temporal Logic of Programs. In *FOCS 1977*, pages 46–57, 1977.
- 20 Markus N. Rabe. *A Temporal Logic Approach to Information-flow Control*. PhD thesis, Saarland University, 2016.
- 21 A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985. doi:10.1145/3828.3837.

## 30:14 The First-Order Logic of Hyperproperties

- 22 Wolfgang Thomas. Languages, automata, and logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Vol. 3*, pages 389–455. Springer, 1997. doi:10.1007/978-3-642-59126-6.
- 23 Wolfgang Thomas. Path logics with synchronization. In Kamal Lodaya, Madhavan Mukund, and R. Ramanujam, editors, *Perspectives in Concurrency Theory*, pages 469–481. IARCS-Universities, Universities Press, 2009.



# Improving and Extending the Testing of Distributions for Shape-Restricted Properties

Eldar Fischer<sup>1</sup>, Oded Lachish<sup>2</sup>, and Yadu Vasudev<sup>\*3</sup>

- 1 Faculty of Computer Science, Israel Institute of Technology (Technion), Haifa, Israel  
eldar@cs.technion.ac.il
- 2 Birkbeck, University of London, London, UK  
oded@dcs.bbk.ac.uk
- 3 Fakultät für Informatik, TU Dortmund, Dortmund, Germany  
yaduvasev@gmail.com

---

## Abstract

Distribution testing deals with what information can be deduced about an unknown distribution over  $\{1, \dots, n\}$ , where the algorithm is only allowed to obtain a relatively small number of independent samples from the distribution. In the extended conditional sampling model, the algorithm is also allowed to obtain samples from the restriction of the original distribution on subsets of  $\{1, \dots, n\}$ .

In 2015, Canonne, Diakonikolas, Gouleakis and Rubinfeld unified several previous results, and showed that for any property of distributions satisfying a “decomposability” criterion, there exists an algorithm (in the basic model) that can distinguish with high probability distributions satisfying the property from distributions that are far from it in variation distance.

We present here a more efficient yet simpler algorithm for the basic model, as well as very efficient algorithms for the conditional model, which until now was not investigated under the umbrella of decomposable properties. Additionally, we provide an algorithm for the conditional model that handles a much larger class of properties.

Our core mechanism is a way of efficiently producing an interval-partition of  $\{1, \dots, n\}$  that satisfies a “fine-grain” quality. We show that with such a partition at hand we can directly move forward with testing individual intervals, instead of first searching for the “correct” partition of  $\{1, \dots, n\}$ .

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, G.3 Probability and Statistics

**Keywords and phrases** conditional sampling, distribution testing, property testing, statistics

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.31

## 1 Introduction

### 1.1 Historical background

In most computational problems that arise from modeling real-world situations, we are required to analyze large amounts of data to decide if it satisfies a fixed property. The amount of data involved is usually too large for reading it in its entirety, both with respect to time and storage. In such situations, it is natural to ask for algorithms that can sample

---

\* This work was done when the author was a postdoctoral fellow at Israel Institute of Technology (Technion), Haifa, Israel.



points from the data and obtain a significant estimate for the property of interest. The area of *property testing* addresses this issue by studying algorithms that look at a small part of the data, and then decide if the object that generated the data has the property or is far (according to some metric) from having the property.

There has been a long line of research, especially in statistics, where the underlying object from which we obtain the data is modeled as a probability distribution. Here the algorithm is only allowed to ask for independent samples from the distribution, and has to base its decision on them. If the support of the underlying probability distribution is large, it is not practical to approximate the entire distribution. Thus, it is natural to study this problem in the context of property testing.

The specific sub-area of property testing that is dedicated to the study of distributions is called *distribution testing*. There, the input is a probability distribution (in this paper the domain is the set  $\{1, 2, \dots, n\}$ ) and the objective is to distinguish whether the distribution has a certain property, such as uniformity or monotonicity, or is far in  $\ell_1$  distance from it. See [7] for a survey about the realm of distribution testing.

Testing properties of distributions was studied by Batu et al. in [5], where they gave a sublinear query algorithm for testing closeness of distributions supported over the set  $\{1, 2, \dots, n\}$ . They extended the idea of collision counting, which was implicitly used for uniformity testing in the work of Goldreich and Ron ([15]). Consequently, various properties of probability distributions were studied, like testing identity with a known distribution ([4, 19, 2, 12]), testing independence of a distribution over a product space ([4, 2]), and testing  $k$ -wise independence ([3]).

In recent years, distribution testing has been extended beyond the classical model. A new model called the *conditional sampling model* was introduced. It first appeared independently in [9] and [10]. In the conditional sampling model, the algorithm queries the input distribution  $\mu$  with a set  $S \subseteq \{1, 2, \dots, n\}$ , and gets an index sampled according to  $\mu$  conditioned on the set  $S$ . Notice that if  $S = \{1, 2, \dots, n\}$ , then this is exactly like in the standard model. The conditional sampling model allows adaptive querying of  $\mu$ , since we can choose the set  $S$  based on the indexes sampled until now. Chakraborty et al. ([10]) and Canonne et al. ([9]) showed that testing uniformity can be done with a number of queries not depending on  $n$  (the latter presenting an optimal test), and investigated the testing of other properties of distributions. In [10], it is also shown that uniformity can be tested with  $\text{poly}(\log n)$  conditional samples by a *non-adaptive* algorithm. In this work, we study distribution testing in the standard (unconditional) sampling model, as well as in the conditional model.

A line of work which is central to our paper, is the testing of distributions for *structure*. The objective is to test whether a given distribution has some structural properties like being monotone ([6]), being a  $k$ -histogram ([16, 12]), or being log-concave ([2]). Canonne et al. ([8]) unified these results to show that if a property of distributions has certain structural characteristics, then membership in the property can be tested efficiently using samples from the distribution. More precisely, they introduced the notion of  *$L$ -decomposable* distributions as a way to unify various algorithms for testing distributions for structure. Informally, an  $L$ -decomposable distribution  $\mu$  supported over  $\{1, 2, \dots, n\}$  is one that has an interval partition  $\mathcal{I}$  of  $\{1, 2, \dots, n\}$  of size bounded by  $L$ , such that for every interval  $I$ , either the weight of  $\mu$  on it is small or the distribution over the interval is close to uniform. A property  $\mathcal{C}$  of distributions is  $L$ -decomposable if every distribution  $\mu \in \mathcal{C}$  is  $L$ -decomposable ( $L$  is allowed to depend on  $n$ ). This generalizes various properties of distributions like being monotone, unimodal, log-concave etc. In this setting, their result for a set of distributions  $\mathcal{C}$  supported over  $\{1, 2, \dots, n\}$  translates to the following: if every distribution  $\mu$  from  $\mathcal{C}$  is

$L$ -decomposable, then there is an efficient algorithm for testing whether a given distribution belongs to the property  $\mathcal{C}$ .

To achieve their results, Canonne et al. ([8]) show that if a distribution  $\mu$  supported over  $[n]$  is  $L$ -decomposable, then it is  $O(L \log n)$ -decomposable where the intervals are of the form  $[j2^i + 1, (j + 1)2^i]$ . This presents a natural approach of computing the interval partition in a recursive manner, by bisecting an interval if it has a large probability weight and is not close to uniform. Once they get an interval partition, they learn the “flattening” of the distribution over this partition, and check if this distribution is close to the property  $\mathcal{C}$ . The term “flattening” refers to the distribution resulting from making  $\mu$  conditioned on any interval of the partition to be uniform. When applied to a partition corresponding to a decomposition of the distribution, the learned flattening is also close to the original distribution. Because of this, in the case where there is a promise that  $\mu$  is  $L$ -decomposable, the above can be viewed as a learning algorithm, where they obtain an explicit distribution that is close to  $\mu$ . Without the promise it can be viewed as an agnostic learning algorithm. For further elaboration of this connection see [11].

## 1.2 Results and techniques

In this paper, we extend the body of knowledge about testing  $L$ -decomposable properties. We improve upon the previously known bound on the sample complexity, and give much better bounds when conditional samples are allowed. Additionally, for the conditional model, we provide a test for a broader family of properties, that we call *atlas-characterizable* properties. Atlas-characterizable properties include the family of *symmetric* properties, of which a subset is studied in [20] in the unconditional context, and [10] studies them in the conditional model.

Our approach differs from that of [8] in the manner in which we obtain the interval partition. We show that a partition where most intervals that are not singletons have small probability weight is sufficient to learn the distribution  $\mu$ , even though it is not the original  $L$ -decomposition of  $\mu$ . We show that if a distribution  $\mu$  is  $L$ -decomposable, then the “flattening” of  $\mu$  with respect to this partition is close to  $\mu$ . It turns out that such a partition can be obtained in “one shot” without resorting to a recursive search procedure.

We obtain a partition as above using a method of *partition pulling* that we develop here. Informally, a pulled partition is obtained by sampling indexes from  $\mu$ , and taking the partition induced by the samples in the following way: each sampled index is a singleton interval, and the rest of the partition is composed of the maximal intervals between sampled indexes. Apart from the obvious simplicity of this procedure, it also has the advantage of providing a partition with a significantly smaller number of intervals, linear in  $L$  for a fixed  $\epsilon$ , and with no dependency on  $n$  unless  $L$  itself depends on it. This makes our algorithm more efficient in query complexity than the one of [8] in the unconditional sampling model, and leads to a dramatically small sampling complexity in the (adaptive) conditional model.

Another feature of the partition pulling method is that it provides a partition with small weight intervals also when the distribution is not  $L$ -decomposable. This allows us to use the partition in a different manner later on, in the algorithm for testing atlas-characterizable properties using conditional samples.

The main common ground between our approach for  $L$ -decomposable properties and that of [8] is the method of testing by implicit learning, as defined formally in [13] (see [18]). In particular, the results also provide a means to learn a distribution close to  $\mu$  if  $\mu$  satisfies the tested property. We also provide a test under the conditional query model for the extended class of atlas-characterizable properties that we define below, which generalizes both decomposable and symmetric properties. A learning algorithm for this class is not provided; only an “atlas” of the input distribution rather than the distribution itself is learned.

■ **Table 1** Summary of our results.

	Result	Known lower bound
<b><i>L</i>-decomposable</b>	(testing and learning)	
Unconditional	$\sqrt{nL}/\text{poly}(\epsilon)$	$\Omega(\sqrt{n}/\epsilon^2)$ for $L = 1$ [17]
Adaptive conditional	$L/\text{poly}(\epsilon)$	$\Omega(L)$ for some fixed $\epsilon$ [10]
Non-adaptive conditional	$L \cdot \text{poly}(\log n, 1/\epsilon)$	$\Omega(\log n)$ for $L = 1$ and some fixed $\epsilon$ [1]
<b><i>k</i>-characterized by atlases</b>	(testing)	
Adaptive conditional	$k \cdot \text{poly}(\log n, 1/\epsilon)$	$\Omega(\sqrt{\log \log n})$ for $k = 1$ , and some fixed $\epsilon$ [10]

Our result for unconditional testing (Theorem 27) gives a  $\sqrt{nL}/\text{poly}(\epsilon)$  query algorithm in the standard (unconditional) sampling model for testing an  $L$ -decomposable property of distributions. Our method of finding a good partition for  $\mu$  using pulled partitions, that we explained above, avoids the  $\log n$  factor present in Theorem 3.3 of [8]. The same method enables us to extend our results to the conditional query model, which we present for both adaptive and non-adaptive algorithms. Table 1 summarizes our results and known lower bounds<sup>1</sup>. We note that the hidden dependencies on  $\epsilon$  in the theorems we present are not optimized. The optimized versions will appear in the journal version of this paper.

## 2 Preliminaries

We study the problem of testing properties of probability distributions supported over  $[n]$  (the set  $\{1, \dots, n\}$ ), when given samples from the distribution. For two distributions  $\mu$  and  $\chi$ , we say that  $\mu$  is  $\epsilon$ -far from  $\chi$  if they are far in the  $\ell_1$  norm, that is,  $d(\mu, \chi) = \sum_{i \in [n]} |\mu(i) - \chi(i)| > \epsilon$ . For a property of distributions  $\mathcal{C}$ , we say that  $\mu$  is  $\epsilon$ -far from  $\mathcal{C}$  if for all  $\chi \in \mathcal{C}$ ,  $d(\mu, \chi) > \epsilon$ .

Outside the  $\ell_1$  norm between distributions, we also use the  $\ell_\infty$  norm,  $\|\mu - \chi\|_\infty = \max_{i \in [n]} |\mu(i) - \chi(i)|$ , the following measure for uniformity and, implicitly throughout, an observation that is directly implied from this definition.

► **Definition 1.** For a distribution  $\mu$  over a domain  $I$ , we define the *bias* of  $\mu$  to be  $\text{bias}(\mu) = \max_{i \in I} \mu(i) / \min_{i \in I} \mu(i) - 1$ .

► **Observation 2.** For any two distributions  $\mu$  and  $\chi$  over a domain  $I$  of size  $m$ ,  $d(\mu, \chi) \leq m \|\mu - \chi\|_\infty$ . Also,  $\|\mu - \mathcal{U}_I\|_\infty \leq \frac{1}{m} \text{bias}(\mu)$ , where  $\mathcal{U}_I$  is the uniform distribution over  $I$ .

We study the problem, both in the standard model, where the algorithm is given indexes sampled from the distribution, as well as in the model of conditional samples. The conditional model was first studied in the independent works of Chakraborty et al. ([10]) and Canonne et al ([9]). We first give the definition of a conditional oracle for a distribution  $\mu$ .

<sup>1</sup> The lower bounds for unconditional and non-adaptive conditional testing of  $L$ -decomposable properties with  $L = 1$  are exactly the lower bounds for uniformity testing; the lower bound for adaptive conditional testing follows easily from the proved existence of properties that have no sub-linear complexity adaptive conditional tests; finally, the lower bound for properties  $k$ -characterized by atlases with  $k = 1$  is just a bound for a symmetric property constructed there. About the last one, we conjecture that there exist properties with much higher lower bounds.

**Input:** Distribution  $\mu$  supported over  $[n]$ , parameters  $\eta > 0$  (finesness) and  $\delta > 0$  (error probability)

- 1 Take  $m = \frac{3}{\eta} \log\left(\frac{3}{\eta\delta}\right)$  unconditional samples from  $\mu$
- 2 Arrange the indices sampled in increasing order  $i_1 < i_2 < \dots < i_r$  without repetition and set  $i_0 = 0$
- 3 **for** each  $j \in [r]$  **do**
- 4     **if**  $i_j > i_{j-1} + 1$  **then** add the interval  $\{i_{j-1} + 1, \dots, i_j - 1\}$  to  $\mathcal{I}$ ;
- 5     Add the singleton interval  $\{i_j\}$  to  $\mathcal{I}$
- 6 **if**  $i_r < n$  **then** add the interval  $\{i_r + 1, \dots, n\}$  to  $\mathcal{I}$ ;
- 7 **return**  $\mathcal{I}$

**Algorithm 1:** Pulling an  $\eta$ -fine partition.

► **Definition 3** (conditional oracle). A conditional oracle to a distribution  $\mu$  supported over  $[n]$  is a black-box that takes as input a set  $A \subseteq [n]$ , samples a point  $i \in A$  with probability  $\mu(i) / \sum_{j \in A} \mu(j)$ , and returns  $i$ . If  $\sum_{j \in A} \mu(j) = 0$ , then it chooses  $i \in A$  uniformly at random.

► **Remark.** The behaviour of the conditional oracle on sets  $A$  with  $\mu(A) = 0$  is as per the model of Chakraborty et al. [10]. However, upper bounds in this model also hold in the model of Canonne et al. [9], and most lower bounds can be easily converted to it.

Now we define adaptive conditional distribution testing algorithms. The definition of their non-adaptive version, which we will also analyze, appears in [14].

► **Definition 4.** An *adaptive* conditional distribution testing algorithm for a property of distributions  $\mathcal{C}$ , with parameters  $\epsilon, \delta > 0$ , and  $n \in \mathbb{N}$ , with query complexity  $q(\epsilon, \delta, n)$ , is a randomized algorithm with access to a conditional oracle of a distribution  $\mu$  with the following properties:

- For each  $i \in [q]$ , at the  $i^{\text{th}}$  phase, the algorithm generates a set  $A_i \subseteq [n]$ , based on  $j_1, j_2, \dots, j_{i-1}$  and its internal coin tosses, and calls the conditional oracle with  $A_i$  to receive an element  $j_i$ , drawn independently of  $j_1, j_2, \dots, j_{i-1}$ .
- Based on the received elements  $j_1, j_2, \dots, j_q$  and its internal coin tosses, the algorithm accepts or rejects the distribution  $\mu$ .

If  $\mu \in \mathcal{C}$ , then the algorithm accepts with probability at least  $1 - \delta$ , and if  $\mu$  is  $\epsilon$ -far from  $\mathcal{C}$ , then the algorithm rejects with probability at least  $1 - \delta$ .

### 3 Fine partitions and how to pull them

We define the notion of  $\eta$ -fine partitions of a distribution  $\mu$  supported over  $[n]$ , which are central to all our algorithms.

► **Definition 5** ( $\eta$ -fine interval partition). Given a distribution  $\mu$  over  $[n]$ , an  $\eta$ -fine interval partition of  $\mu$  is an interval-partition  $\mathcal{I} = (I_1, I_2, \dots, I_r)$  of  $[n]$  such that for all  $j \in [r]$ ,  $\mu(I_j) \leq \eta$ , excepting the case  $|I_j| = 1$ . The *length*  $|\mathcal{I}|$  of an interval partition  $\mathcal{I}$  is the number of intervals in it.

Algorithm 1 is the pulling mechanism. The idea is to take independent unconditional samples from  $\mu$ , make them into singleton intervals in our interval-partition  $\mathcal{I}$ , and then take the intervals between these samples as the remaining intervals in  $\mathcal{I}$ .

**Input:** Distribution  $\mu$  supported over  $[n]$ , parameters  $\eta, \gamma > 0$  (fineness) and  $\delta > 0$  (error probability)

- 1 Take  $m = \frac{3}{\eta} \log\left(\frac{5}{\gamma\delta}\right)$  unconditional samples from  $\mu$
- 2 Perform Step 2 through Step 6 of Algorithm 1.
- 3 **return**  $\mathcal{I}$

**Algorithm 2:** Pulling an  $(\eta, \gamma)$ -fine partition.

► **Lemma 6.** *Let  $\mu$  be a distribution that is supported over  $[n]$ , and  $\eta, \delta > 0$ , and suppose that these are fed to Algorithm 1. Then, with probability at least  $1 - \delta$ , the set of intervals  $\mathcal{I}$  returned by Algorithm 1 is an  $\eta$ -fine interval partition of  $\mu$  of length  $O\left(\frac{1}{\eta} \log\left(\frac{1}{\eta\delta}\right)\right)$ .*

**Proof.** Let  $\mathcal{I}$  the set of intervals returned by Algorithm 1. The guarantee on the length of  $\mathcal{I}$  follows from the number of samples taken in Step 1, noting that  $|\mathcal{I}| \leq 2r - 1 = O(m)$ .

Let  $\mathcal{J}$  be a maximal set of pairwise disjoint minimal intervals  $I$  in  $[n]$ , such that  $\mu(I) \geq \eta/3$  for every interval  $I \in \mathcal{J}$ . Note that every  $i$  for which  $\mu(i) \geq \eta/3$  necessarily appears as a singleton interval  $\{i\} \in \mathcal{J}$ . Also clearly  $|\mathcal{J}| \leq 3/\eta$ .

We shall first show that if an interval  $I'$  is such that  $\mu(I') \geq \eta$ , then it fully contains some interval  $I \in \mathcal{J}$ . Then, we shall show that, with probability at least  $1 - \delta$ , the samples taken in Step 1 include an index from every interval  $I \in \mathcal{J}$ . By Steps 2 to 6 of the algorithm and the above, this implies the statement of the lemma.

Let  $I'$  be an interval such that  $\mu(I') \geq \eta$ , and assume on the contrary that it contains no interval from  $\mathcal{J}$ . Clearly it may intersect without containing at most two intervals  $I_l, I_r \in \mathcal{J}$ . Also,  $\mu(I' \cap I_l) < \eta/3$  because otherwise we could have replaced  $I_l$  with  $I' \cap I_l$  in  $\mathcal{J}$ , and the same holds for  $\mu(I' \cap I_r)$ . But this means that  $\mu(I' \setminus (I_l \cup I_r)) > \eta/3$ , and so we could have added  $I' \setminus (I_l \cup I_r)$  to  $\mathcal{J}$ , again a contradiction.

Let  $I \in \mathcal{J}$ . The probability that an index from  $I$  is not sampled is at most  $(1 - \eta/3)^{3 \log(3/\eta\delta)/\eta} \leq \delta\eta/3$ . By a union bound over all  $I \in \mathcal{J}$ , with probability at least  $1 - \delta$ , the samples taken in Step 1 include an index from every interval in  $\mathcal{J}$ . ◀

The following is a definition of a variation of a fine partition, where we allow some intervals of small total weight to violate the original requirements.

► **Definition 7** ( $(\eta, \gamma)$ -fine partitions). Given a distribution  $\mu$  over  $[n]$ , an  $(\eta, \gamma)$ -fine interval partition is an interval partition  $\mathcal{I} = (I_1, I_2, \dots, I_r)$  such that  $\sum_{I \in \mathcal{H}_{\mathcal{I}}} \mu(I) \leq \gamma$ , where  $\mathcal{H}_{\mathcal{I}}$  is the set of *violating intervals*  $\{I \in \mathcal{I} : \mu(I) > \eta, |I| > 1\}$ .

In our applications,  $\gamma$  will be larger than  $\eta$  by a factor of  $L$ , which would allow us through Algorithm 2 to avoid having additional  $\log L$  factors in our expressions for the unconditional and the adaptive tests.

► **Lemma 8.** *Let  $\mu$  be a distribution that is supported over  $[n]$ , and  $\gamma, \eta, \delta > 0$ , and suppose that these are fed to Algorithm 2. Then, with probability at least  $1 - \delta$ , the set of intervals  $\mathcal{I}$  returned by Algorithm 2 is an  $(\eta, \gamma)$ -fine interval partition of  $\mu$  of length  $O\left(\frac{1}{\eta} \log\left(\frac{1}{\gamma\delta}\right)\right)$ .*

The proof of this lemma is based on Lemma 6 and appears in [14].

## 4 Handling decomposable distributions

The notion of  $L$ -decomposable distributions was defined and studied in [8]. They showed that a large class of properties, such as monotonicity and log-concavity, are  $L$ -decomposable. We now formally define  $L$ -decomposable distributions and properties, as given in [8].

► **Definition 9** ( $(\gamma, L)$ -decomposable distributions [8]). For an integer  $L$ , a distribution  $\mu$  supported over  $[n]$  is  $(\gamma, L)$ -decomposable, if there exists an interval partition  $\mathcal{I} = (I_1, I_2, \dots, I_\ell)$  of  $[n]$ , where  $\ell \leq L$ , such that for all  $j \in [\ell]$ , at least one of the following holds.

1.  $\mu(I_j) \leq \frac{\gamma}{L}$ .
2.  $\max_{i \in I_j} \mu(i) \leq (1 + \gamma) \min_{i \in I_j} \mu(i)$ .

The second condition in the definition of a  $(\gamma, L)$ -decomposable distribution is identical to saying that  $\text{bias}(\mu \upharpoonright_{I_j}) \leq \gamma$ . An  $L$ -decomposable property is now defined in terms of all its members being decomposable distributions.

► **Definition 10** ( $L$ -decomposable properties, [8]). For a function  $L : (0, 1] \times \mathbb{N} \rightarrow \mathbb{N}$ , we say that a property of distributions  $\mathcal{C}$  is  $L$ -decomposable, if for every  $\gamma > 0$ , and  $\mu \in \mathcal{C}$  supported over  $[n]$ ,  $\mu$  is  $(\gamma, L(\gamma, n))$ -decomposable.

Recall that part of the algorithm for learning such distributions is finding (through pulling) what we referred to as a fine partition. Such a partition may still have intervals where the conditional distribution over them is far from uniform. However, we shall show that for  $L$ -decomposable distributions, the total weight of such “bad” intervals is not high.

The next lemma shows that every fine partition of an  $(\gamma, L)$ -decomposable distribution has only a small weight concentrated on “non-uniform” intervals, and thus it will be sufficient to deal with the “uniform” intervals.

► **Lemma 11.** *Let  $\mu$  be a distribution supported over  $[n]$  which is  $(\gamma, L)$ -decomposable. For every  $\gamma/L$ -fine interval partition  $\mathcal{I}' = (I'_1, I'_2, \dots, I'_r)$  of  $\mu$ , the following holds:  $\sum_{j \in [r]: \text{bias}(\mu \upharpoonright_{I'_j}) > \gamma} \mu(I'_j) \leq 2\gamma$ .*

**Proof.** Let  $\mathcal{I} = (I_1, I_2, \dots, I_\ell)$  be the  $L$ -decomposition of  $\mu$ , where  $\ell \leq L$ . Let  $\mathcal{I}' = (I'_1, I'_2, \dots, I'_r)$  be an interval partition of  $[n]$  such that for all  $j \in [r]$ ,  $\mu(I'_j) \leq \gamma/L$  or  $|I'_j| = 1$ .

Any interval  $I'_j$  for which  $\text{bias}(\mu \upharpoonright_{I'_j}) > \gamma$ , is either completely inside an interval  $I_k$  such that  $\mu(I_k) \leq \gamma/L$ , or intersects more than one interval (and in particular  $|I'_j| > 1$ ). There are at most  $L - 1$  intervals in  $\mathcal{I}'$  that intersect more than one interval in  $\mathcal{I}$ . The sum of the weights of all such intervals is at most  $\gamma$ .

For any interval  $I_k$  of  $\mathcal{I}$  such that  $\mu(I_k) \leq \gamma/L$ , the sum of the weights of intervals from  $\mathcal{I}'$  that lie completely inside  $I_k$  is at most  $\gamma/L$ . Thus, the total weight of all such intervals is bounded by  $\gamma$ . Therefore, the sum of the weights of intervals  $I'_j$  such that  $\text{bias}(\mu \upharpoonright_{I'_j}) > \gamma$  is at most  $2\gamma$ . ◀

In order to get better bounds, we will use the counterpart of this lemma for the more general (two-parameter) notion of a fine partition.

► **Lemma 12.** *Let  $\mu$  be a distribution supported over  $[n]$  which is  $(\gamma, L)$ -decomposable. For every  $(\gamma/L, \gamma)$ -fine interval partition  $\mathcal{I}' = (I'_1, I'_2, \dots, I'_r)$  of  $\mu$ , the following holds:  $\sum_{j \in [r]: \text{bias}(\mu \upharpoonright_{I'_j}) > \gamma} \mu(I'_j) \leq 3\gamma$ .*

**Proof.** Let  $\mathcal{I} = (I_1, I_2, \dots, I_\ell)$  be the  $L$ -decomposition of  $\mu$ , where  $\ell \leq L$ . Let  $\mathcal{I}' = (I'_1, I'_2, \dots, I'_r)$  be an interval partition of  $[n]$  such that for a set  $\mathcal{H}_{\mathcal{I}'}$  of total weight at most  $\gamma$ , for all  $I'_j \in \mathcal{I}' \setminus \mathcal{H}_{\mathcal{I}'}$ ,  $\mu(I'_j) \leq \gamma/L$  or  $|I'_j| = 1$ .

Exactly as in the proof of Lemma 11, the total weight of intervals  $I'_j \in \mathcal{I}' \setminus \mathcal{H}_{\mathcal{I}'}$  for which  $\text{bias}(\mu \upharpoonright_{I'_j}) > \gamma$  is at most  $2\gamma$ . In the worst case, all intervals in  $\mathcal{H}_{\mathcal{I}'}$  are also such that  $\text{bias}(\mu \upharpoonright_{I'_j}) > \gamma$ , adding at most  $\gamma$  to the total weight of such intervals. ◀

As previously mentioned, we are not learning the actual distribution but a “flattening” thereof. We next formally define the flattening of a distribution  $\mu$  with respect to an interval partition  $\mathcal{I}$ . Afterwards we shall describe its advantages and how it can be learned.

► **Definition 13.** Given a distribution  $\mu$  supported over  $[n]$  and a partition  $\mathcal{I} = (I_1, I_2, \dots, I_\ell)$ , of  $[n]$  to intervals, the *flattening* of  $\mu$  with respect to  $\mathcal{I}$  is a distribution  $\mu_{\mathcal{I}}$ , supported over  $[n]$ , such that for  $i \in I_j$ ,  $\mu_{\mathcal{I}}(i) = \mu(I_j)/|I_j|$ .

The following lemma shows that the flattening of any distribution  $\mu$ , with respect to any interval partition that has only small weight on intervals far from uniform, is close to  $\mu$ .

► **Lemma 14.** Let  $\mu$  be a distribution supported on  $[n]$ , and let  $\mathcal{I} = (I_1, I_2, \dots, I_r)$  be an interval partition of  $\mu$  such that  $\sum_{j \in [r]: d(\mu \upharpoonright_{I_j}, \mathcal{U}_{I_j}) \geq \gamma} \mu(I_j) \leq \eta$ . Then  $d(\mu, \mu_{\mathcal{I}}) \leq \gamma + 2\eta$ .

The proof of this lemma appears in [14]. The good thing about a flattening (for an interval partition of small length) is that it can be efficiently learned. For this we first make a technical definition and note a trivial observation, whose proof follows immediately from the definition:

► **Definition 15 (Coarsening).** Given  $\mu$  and  $\mathcal{I}$ , where  $|\mathcal{I}| = \ell$ , we define the *coarsening* of  $\mu$  according to  $\mathcal{I}$  the distribution  $\hat{\mu}_{\mathcal{I}}$  over  $[\ell]$  as by  $\hat{\mu}_{\mathcal{I}}(j) = \mu(I_j)$  for all  $j \in [\ell]$ .

► **Observation 16.** Given a distribution  $\hat{\mu}_{\mathcal{I}}$  over  $[\ell]$ , define  $\mu_{\mathcal{I}}$  over  $[n]$  by  $\mu_{\mathcal{I}}(i) = \hat{\mu}_{\mathcal{I}}(j_i)/|I_{j_i}|$ , where  $j_i$  is the index satisfying  $i \in I_{j_i}$ . This is a distribution, and for any two distributions  $\hat{\mu}_{\mathcal{I}}$  and  $\hat{\chi}_{\mathcal{I}}$  we have  $d(\mu_{\mathcal{I}}, \chi_{\mathcal{I}}) = d(\hat{\mu}_{\mathcal{I}}, \hat{\chi}_{\mathcal{I}})$ . Moreover, if  $\hat{\mu}_{\mathcal{I}}$  is a coarsening of a distribution  $\mu$  over  $[n]$ , then  $\mu_{\mathcal{I}}$  is the respective flattening of  $\mu$ .

The following lemma, which is proved in [14], shows how learning can be achieved. We will ultimately use this in conjunction with Lemma 14 as a means to learn a whole distribution through its flattening.

► **Lemma 17.** Given a distribution  $\mu$  supported over  $[n]$  and an interval partition  $\mathcal{I} = (I_1, I_2, \dots, I_\ell)$ , using  $\frac{2(\ell + \log(2/\delta))}{\epsilon^2}$  samples, we can obtain an explicit distribution  $\mu'_{\mathcal{I}}$ , supported over  $[n]$ , such that, with probability at least  $1 - \delta$ ,  $d(\mu_{\mathcal{I}}, \mu'_{\mathcal{I}}) \leq \epsilon$ .

## 5 Weakly tolerant interval uniformity tests

To unify our treatment of learning and testing with respect to  $L$ -decomposable properties to all three models (unconditional, adaptive-conditional and non-adaptive-conditional), we first define what it means to test a distribution  $\mu$  for uniformity over an interval  $I \subseteq [n]$ . The following definition is technical in nature, but it is what we need to be used as a building block for our learning and testing algorithms.

► **Definition 18 (Weakly tolerant interval tester).** A *weakly tolerant interval tester* is an algorithm  $\mathbb{T}$  that takes as input a distribution  $\mu$  over  $[n]$ , an interval  $I \subseteq [n]$ , a maximum size parameter  $m$ , a minimum weight parameter  $\gamma$ , an approximation parameter  $\epsilon$  and an error parameter  $\delta$ , and satisfies the following.

1. If  $|I| \leq m$ ,  $\mu(I) \geq \gamma$ , and  $\text{bias}(\mu \upharpoonright_I) \leq \epsilon/100$ , then the algorithm accepts with probability at least  $1 - \delta$ .
2. If  $|I| \leq m$ ,  $\mu(I) \geq \gamma$ , and  $d(\mu \upharpoonright_I, \mathcal{U}_I) > \epsilon$ , then the algorithm rejects with probability at least  $1 - \delta$ .

In all other cases, the algorithm may accept or reject with arbitrary probability.



**Input:** A distribution  $\mu$  supported over  $[n]$ , parameters  $c, r$ , an interval partition  $\mathcal{I}$  satisfying  $|\mathcal{I}| \leq r$ , parameters  $\epsilon, \delta > 0$ , a weakly tolerant interval uniformity tester  $\mathbb{T}$  taking input values  $(\mu, I, m, \gamma, \epsilon, \delta)$ .

- 1 **for**  $s = 20 \log(1/\delta)/\epsilon$  **times do**
- 2     Take an unconditional sample from  $\mu$  and let  $I \in \mathcal{I}$  be the interval that contains it
- 3     Use the tester  $\mathbb{T}$  with input values  $(\mu, I, n/c, \epsilon/r, \epsilon, \delta/2s)$
- 4     **if** *test rejects* **then** add  $I$  to  $\mathcal{B}$ ;
- 5 **if**  $|\mathcal{B}| > 4\epsilon s$  **then reject else accept**

**Algorithm 3:** Assessing a partition.

For our purposes we will use three weakly tolerant interval testers, one for every model. These are summed up in the following lemma. They mostly build on previous work on uniformity testing; the proofs are found in [14].

► **Lemma 19.** *For the input  $(\mu, I, m, \gamma, \epsilon, \delta)$ , there exist these weakly tolerant interval testers:*

1. *A tester using  $O(\sqrt{m} \log(1/\delta)/\gamma\epsilon^2)$  unconditional samples from  $\mu$ .*
2. *A tester that adaptively takes  $\log(1/\delta) \text{poly}(\log(1/\epsilon))/\epsilon^2$  conditional samples from  $\mu$ .*
3. *A non-adaptive tester that takes  $\text{poly}(\log n, 1/\epsilon) \log(1/\delta)/\gamma$  conditional samples from  $\mu$ , where just the decision depends on  $I$  and the queries depend only on the other parameters.*

Note that the independence of the queries of  $I$  in Item 3 is important in the sequel to keep the algorithm using it a non-adaptive one.

## 6 Assessing an interval partition

Through either Lemma 6 or Lemma 8 we can construct a fine partition, and then through either Lemma 11 or Lemma 12 respectively we know that if  $\mu$  is decomposable, then most of the weight is concentrated on intervals with a small bias. However, eventually we would like a test that works for decomposable and non-decomposable distributions alike. For this we need a way to assess an interval partition as to whether it is indeed suitable for learning a distribution. This is done through a weighted sampling of intervals, for which we employ a weakly tolerant tester. The following is the formal description, given as Algorithm 3.

To analyze it, first, for a fine interval partition, we bound the total weight of intervals where the weakly tolerant tester is not guaranteed a small error probability; recall that  $\mathbb{T}$  as used in Step 3 guarantees a correct output only for an interval  $I$  satisfying  $\mu(I) \geq \epsilon/r$  and  $|I| \leq n/c$ .

► **Observation 20.** *Define  $\mathcal{N}_{\mathcal{I}} = \{I \in \mathcal{I} : |I| > n/c \text{ or } \mu(I) < \epsilon/r\}$ . If  $\mathcal{I}$  is  $(\eta, \gamma)$ -fine, where  $c\eta + \gamma \leq \epsilon$ , then  $\mu(\bigcup_{I \in \mathcal{N}_{\mathcal{I}}} I) \leq 2\epsilon$ .*

The proof of this observation appears in [14]. The following “completeness” lemma states that the typical case for a fine partition of a decomposable distribution, i.e. the case where most intervals exhibit a small bias, is correctly detected.

► **Lemma 21.** *Suppose that  $\mathcal{I}$  is  $(\eta, \gamma)$ -fine, where  $c\eta + \gamma \leq \epsilon$ . Define  $\mathcal{G}_{\mathcal{I}} = \{i : \mathcal{I} : \text{bias}(\mu \upharpoonright_I) \leq \epsilon/100\}$ . If  $\mu(\bigcup_{I \in \mathcal{G}_{\mathcal{I}}} I) \geq 1 - \epsilon$ , then Algorithm 3 accepts with probability at least  $1 - \delta$ .*

The proof of this lemma appears in [14]. The following “soundness” lemma states that if too much weight is concentrated on intervals where  $\mu$  is far from uniform in the  $\ell_1$  distance,

- Input:** Distribution  $\mu$  supported over  $[n]$ , parameters  $L$  (decomposability),  $\epsilon > 0$  (accuracy), a weakly tolerant interval uniformity tester  $\mathbb{T}$  taking input values  $(\mu, I, m, \gamma, \epsilon, \delta)$
- 1 Use Algorithm 2 with input values  $(\mu, \epsilon/2000L, \epsilon/2000, 1/9)$  to obtain a partition  $\mathcal{I}$  with  $|\mathcal{I}| \leq r = 10^5 L \log(1/\epsilon)/\epsilon$
  - 2 Use Algorithm 3 with input values  $(\mu, L, r, \mathcal{I}, \epsilon/20, 1/9, \mathbb{T})$
  - 3 **if** *Algorithm 3 rejected* **then reject**;
  - 4 Use Lemma 17 with values  $(\mu, \mathcal{I}, \epsilon/10, 1/9)$  to obtain  $\mu'_{\mathcal{I}}$
  - 5 **return**  $\mu'_{\mathcal{I}}$

**Algorithm 4:** Learning an  $L$ -decomposable distribution.

then the algorithm rejects. Later we will show that this is the only situation where  $\mu$  cannot be easily learned through its flattening according to  $\mathcal{I}$ .

► **Lemma 22.** *Suppose that  $\mathcal{I}$  is  $(\eta, \gamma)$ -fine, where  $c\eta + \gamma \leq \epsilon$ . Define  $\mathcal{F}_{\mathcal{I}} = \{i : \mathcal{I} : d(\mu \upharpoonright_I, \mathcal{U}_I) > \epsilon\}$ . If  $\mu(\bigcup_{I \in \mathcal{F}_{\mathcal{I}}} I) \geq 7\epsilon$ , then Algorithm 3 rejects with probability at least  $1 - \delta$ .*

The proof of this lemma appears in [14]. Finally, we present the query complexity of the algorithm. It is presented as generally quadratic in  $\log(1/\delta)$ , but this can be made linear easily by first using the algorithm with  $\delta = 1/3$ , and then repeating it  $O(\log(1/\delta))$  times and taking the majority vote. When we use this lemma later on, both  $r$  and  $c$  will be linear in the decomposability parameter  $L$  for a fixed  $\epsilon$ , and  $\delta$  will be a fixed constant.

► **Lemma 23.** *Algorithm 3 requires  $O(q \log(1/\delta)/\epsilon)$  many samples, where  $q$  is a function of  $n/c, \epsilon/r, \epsilon$  and  $\delta/2s$  that is the number of samples that the invocation of  $\mathbb{T}$  in Step 3 requires.*

*In particular, Algorithm 3 can be implemented either as an unconditional sampling algorithm taking  $r\sqrt{n/c} \log^2(1/\delta)/\text{poly}(\epsilon)$  many samples, an adaptive conditional sampling algorithm taking  $r \log^2(1/\delta)/\text{poly}(\epsilon)$  many samples, or a non-adaptive conditional sampling algorithm taking  $r \log^2(1/\delta) \text{poly}(\log n, 1/\epsilon)$  many samples.*

**Proof.** A single (unconditional) sample is taken each time Step 2 is reached, and all other samples are taken by the invocation of  $\mathbb{T}$  in Step 3. This makes the total number of samples to be  $s(q + 1) = O(q \log(1/\delta)/\epsilon)$ .

The bound for each individual sampling model follows by plugging in Items 1, 2 and 3 of Lemma 19 respectively. For the last one it is important that the tester makes its queries completely independently of  $I$ , as otherwise the algorithm would not have been non-adaptive. ◀

## 7 Learning and testing decomposable distributions and properties

Here we finally put things together to produce a learning algorithm for  $L$ -decomposable distributions. This algorithm is not only guaranteed to learn with high probability a distribution that is decomposable, but is also guaranteed with high probability to not produce a wrong output for any distribution (though it may plainly reject a distribution that is not decomposable). This is presented in Algorithm 4. We present it with a fixed error probability  $2/3$  because this is what we use later on, but it is not hard to move to a general  $\delta$ .

First we show completeness, that the algorithm succeeds for decomposable distributions.

► **Lemma 24.** *If  $\mu$  is  $(\epsilon/2000, L)$ -decomposable, then with probability at least  $2/3$ , Algorithm 4 produces a distribution  $\mu'$  so that  $d(\mu, \mu') \leq \epsilon$ .*

**Proof.** By Lemma 8, with probability at least  $8/9$ , the partition  $\mathcal{I}$  is  $(\epsilon/2000L, \epsilon/2000)$ -fine, which means by Lemma 12 that  $\sum_{j \in [r]: \text{bias}(\mu \upharpoonright_{I'_j}) > \epsilon/2000} \mu(I'_j) \leq 3\epsilon/2000$ . When this occurs, by Lemma 21 with probability at least  $8/9$ , Algorithm 3 will accept and so the algorithm will move past Step 3. In this situation, in particular by Lemma 14 we have that  $d(\mu_{\mathcal{I}}, \mu) \leq 15\epsilon/20$  (in fact this can be bounded much smaller here), and with probability at least  $8/9$  (by Lemma 17) Step 4 provides a distribution that is  $\epsilon/10$ -close to  $\mu_{\mathcal{I}}$  and hence  $\epsilon$ -close to  $\mu$ . ◀

Next we show that the algorithm will, with high probability, not mislead about the distribution, whether it is decomposable or not.

► **Lemma 25.** *For any  $\mu$ , the probability that Algorithm 4 produces (without rejecting) a distribution  $\mu'$  for which  $d(\mu, \mu') > \epsilon$  is bounded by  $2/9$ .*

**Proof.** Consider the interval partition  $\mathcal{I}$ . By Lemma 8, with probability at least  $8/9$ , it is  $(\epsilon/2000L, \epsilon/2000)$ -fine. When this happens, if  $\mathcal{I}$  is such that  $\sum_{j: d(\mu \upharpoonright_{I_j}, \mathcal{M}_{I_j})} \mu(I_j) > 7\epsilon/20$ , then by Lemma 22 with probability at least  $8/9$ , the algorithm will reject in Step 3, and we are done (recall that here a rejection is an allowable outcome).

On the other hand, if  $\mathcal{I}$  is such that  $\sum_{j: d(\mu \upharpoonright_{I_j}, \mathcal{M}_{I_j})} \mu(I_j) \leq 7\epsilon/20$ , then by Lemma 14 we have that  $d(\mu_{\mathcal{I}}, \mu) \leq 15\epsilon/20$ , and with probability at least  $8/9$  (by Lemma 17), Step 4 provides a distribution that is  $\epsilon/10$ -close to  $\mu_{\mathcal{I}}$  and hence  $\epsilon$ -close to  $\mu$ , which is also an allowable outcome. ◀

We now plug in the sample complexity bounds and afterwards summarize all as a theorem.

► **Lemma 26.** *Algorithm 4 requires  $O(L \log(1/\epsilon)/\epsilon + q/\epsilon + L \log(1/\epsilon)/\epsilon^3)$  many samples, where the value  $q = q(n/L, \epsilon^2/10^5 L \log(1/\epsilon), \epsilon/20, \epsilon/2000)$  is a bound on the number of samples that each invocation of  $\mathbb{T}$  inside Algorithm 3 requires.*

*In particular, Algorithm 4 can be implemented either as an unconditional sampling algorithm taking  $\sqrt{nL}/\text{poly}(\epsilon)$  many samples, an adaptive conditional sampling algorithm taking  $L/\text{poly}(\epsilon)$  many samples, or a non-adaptive conditional sampling algorithm taking  $L \text{poly}(\log n, 1/\epsilon)$  many samples.*

**Proof.** The three summands in the general expression follow respectively from the sample complexity calculations of Lemma 8 for Step 1, Lemma 23 for Step 2, and Lemma 17 for Step 4 respectively. Also note that all samples outside Step 2 are unconditional. The bound for each individual sampling model follows from the respective bound stated in Lemma 23. ◀

► **Theorem 27.** *Algorithm 4 is capable of learning an  $(\epsilon/2000, L)$ -decomposable distribution, giving with probability at least  $2/3$ , a distribution that is  $\epsilon$ -close to it, such that for no distribution will it give as output a distribution  $\epsilon$ -far from it with probability more than  $1/3$ .*

*It can be implemented either as an unconditional sampling algorithm taking  $\sqrt{nL}/\text{poly}(\epsilon)$  many samples, an adaptive conditional sampling algorithm taking  $L/\text{poly}(\epsilon)$  many samples, or a non-adaptive conditional sampling algorithm taking  $L \text{poly}(\log n, 1/\epsilon)$  many samples.*

**Proof.** This follows from Lemmas 24, 25 and 26 respectively. ◀

Algorithm 5, next, is a direct application of the above to testing decomposable properties.

► **Theorem 28.** *Algorithm 5 is a test (with error probability  $1/3$ ) for the  $L$ -decomposable property  $\mathcal{C}$ . For  $L = L(\epsilon/4000, n)$ , It can be implemented either as an unconditional sampling algorithm taking  $\sqrt{nL}/\text{poly}(\epsilon)$  many samples, an adaptive conditional sampling algorithm taking  $L/\text{poly}(\epsilon)$  many samples, or a non-adaptive conditional sampling algorithm taking  $L \text{poly}(\log n, 1/\epsilon)$  many samples.*

**Input:** Distribution  $\mu$  supported over  $[n]$ , function  $L : (0, 1] \times \mathbb{N} \rightarrow \mathbb{N}$  (decomposability), parameter  $\epsilon > 0$  (accuracy), an  $L$ -decomposable property  $\mathcal{C}$  of distributions, a weakly tolerant interval uniformity tester  $\mathbb{T}$  taking input values  $(\mu, I, m, \gamma, \epsilon, \delta)$ .

- 1 Use Algorithm 4 with input values  $(\mu, L(\epsilon/4000, n), \epsilon/2, \mathbb{T})$  to obtain  $\mu'$
- 2 **if** Algorithm 4 accepted and  $\mu'$  is  $\epsilon/2$ -close to  $\mathcal{C}$  **then** accept **else** reject

**Algorithm 5:** Testing  $L$ -decomposable properties.

**Proof.** The number and the nature of the samples are determined fully by the application of Algorithm 4 in Step 1, and are thus the same as in Theorem 27. Also by this theorem, for a distribution  $\mu \in \mathcal{C}$ , with probability at least  $2/3$ , an  $\epsilon/2$ -close distribution  $\mu'$  will be produced, and so it will be accepted in Step 2.

Finally, if  $\mu$  is  $\epsilon$ -far from  $\mathcal{C}$ , then with probability at least  $2/3$ , Step 1 will either produce a rejection, or again produce  $\mu'$  that is  $\epsilon/2$ -close to  $\mu$ . In the latter case,  $\mu'$  will be  $\epsilon/2$ -far from  $\mathcal{C}$  by the triangle inequality, and so Step 2 will reject in either case. ◀

## 8 Introducing properties characterized by atlases

In this section, we formally define properties characterized by atlases. It is shown in [14, Section 9], that distributions that are  $L$ -decomposable are, in particular, characterized by atlases. First we start with the definition of an inventory.

► **Definition 29 (Inventory).** Given an interval  $I = [a, b] \subseteq [n]$  and a real-valued function  $\nu : [a, b] \rightarrow [0, 1]$ , the *inventory* of  $\nu$  over  $[a, b]$  is the multiset  $M$  corresponding to  $(\nu(a), \dots, \nu(b))$ .

That is, we keep count of the function values over the interval including repetitions, but ignore their order.

► **Definition 30 (Atlas).** Given a distribution  $\mu$  over  $[n]$ , and an interval partition  $\mathcal{I} = (I_1, \dots, I_k)$  of  $[n]$ , the *atlas*  $\mathcal{A}$  of  $\mu$  over  $\mathcal{I}$  is the ordered pair  $(\mathcal{I}, \mathcal{M})$ , where  $\mathcal{M}$  is the sequence of multisets  $(M_1, \dots, M_k)$  so that  $M_j$  is the inventory of  $\mu$  (considered as a real-valued function) over  $I_j$  for every  $j \in [k]$ . In this setting, we also say that  $\mu$  *conforms* to  $\mathcal{A}$ .

There can be many distributions over  $[n]$  with the same atlas. We also denote by an atlas  $\mathcal{A}$  any ordered pair  $(\mathcal{I}, \mathcal{M})$  where  $\mathcal{I}$  is an interval partition of  $[n]$  and  $\mathcal{M}$  is a sequence of multisets of the same length, so that the total sum of all members of all multisets is 1. It is a simple observation that for every such  $\mathcal{A}$  there exists at least one distribution that conforms to it. The *length* of an atlas  $|\mathcal{A}|$  is defined as the shared length of its interval partition and sequence of multisets. Next we define when a property is characterized by atlases.

► **Definition 31.** For a function  $k : (0, 1] \times \mathbb{N} \rightarrow \mathbb{N}$ , we say that a property of distributions  $\mathcal{C}$  is *k-characterized by atlases* if for every  $n \in \mathbb{N}$  and every  $\epsilon > 0$  we have a set  $\mathbb{A}$  of atlases of lengths bounded by  $k(\epsilon, n)$ , so that every distribution  $\mu$  over  $[n]$  satisfying  $\mathcal{C}$  conforms to some  $\mathcal{A} \in \mathbb{A}$ , while on the other hand no distribution  $\mu$  that conforms to any  $\mathcal{A} \in \mathbb{A}$  is  $\epsilon$ -far from satisfying  $\mathcal{C}$ .

In [14, Section 9], we give some examples of such properties, and show that characterizability is preserved also when switching to a tolerant testing scheme. The following is the main result, whose proof is given in [14, Section 9].

► **Theorem 32.** *If  $\mathcal{C}$  is a property of distributions that is  $k$ -characterized by atlases, then for any  $\epsilon > 0$  there is an adaptive conditional testing algorithm for  $\mathcal{C}$  with query complexity  $k(\epsilon/5, n) \cdot \text{poly}(\log n, 1/\epsilon)$  (and error probability bound  $1/3$ ).*

---

## References

- 1 Jayadev Acharya, Clément L. Canonne, and Gautam Kamath. A chasm between identity and equivalence testing with conditional queries. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, pages 449–466, 2015. doi:10.4230/LIPIcs.APPROX-RANDOM.2015.449.
- 2 Jayadev Acharya, Constantinos Daskalakis, and Gautam Kamath. Optimal testing for properties of distributions. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 3591–3599, 2015.
- 3 Noga Alon, Alexandr Andoni, Tali Kaufman, Kevin Matulef, Ronitt Rubinfeld, and Ning Xie. Testing  $k$ -wise and almost  $k$ -wise independence. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing, STOC'07*, pages 496–505. ACM, 2007. doi:10.1145/1250790.1250863.
- 4 Tugkan Batu, Lance Fortnow, Eldar Fischer, Ravi Kumar, Ronitt Rubinfeld, and Patrick White. Testing random variables for independence and identity. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 442–451, 2001. doi:10.1109/SFCS.2001.959920.
- 5 Tugkan Batu, Lance Fortnow, Ronitt Rubinfeld, Warren D. Smith, and Patrick White. Testing that distributions are close. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 259–269. IEEE Computer Society, 2000. doi:10.1109/SFCS.2000.892113.
- 6 Tugkan Batu, Ravi Kumar, and Ronitt Rubinfeld. Sublinear algorithms for testing monotone and unimodal distributions. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 381–390. ACM, 2004. doi:10.1145/1007352.1007414.
- 7 Clément L. Canonne. A survey on distribution testing: Your data is big. but is it blue? *Electronic Colloquium on Computational Complexity (ECCC)*, 22:63, 2015.
- 8 Clément L. Canonne, Ilias Diakonikolas, Themis Gouleakis, and Ronitt Rubinfeld. Testing shape restrictions of discrete distributions. *CoRR*, abs/1507.03558, 2015.
- 9 Clément L. Canonne, Dana Ron, and Rocco A. Servedio. Testing probability distributions using conditional samples. *SIAM J. Comput.*, 44(3):540–616, 2015. doi:10.1137/130945508.
- 10 Sourav Chakraborty, Eldar Fischer, Yonatan Goldhirsh, and Arie Matsliah. On the power of conditional samples in distribution testing. In *Innovations in Theoretical Computer Science, ITCS'13, Berkeley, CA, USA, January 9-12, 2013*, pages 561–580, 2013. doi:10.1145/2422436.2422497.
- 11 Ilias Diakonikolas. Learning structured distributions. *Handbook of Big Data*, page 267, 2016.
- 12 Ilias Diakonikolas and Daniel M. Kane. A new approach for testing properties of discrete distributions. *CoRR*, abs/1601.05557, 2016.
- 13 Ilias Diakonikolas, Homin K. Lee, Kevin Matulef, Krzysztof Onak, Ronitt Rubinfeld, Rocco A. Servedio, and Andrew Wan. Testing for concise representations. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 549–558, 2007. doi:10.1109/FOCS.2007.70.

- 14 Eldar Fischer, Oded Lachish, and Yadu Vasudev. Improving and extending the testing of distributions for shape-restricted properties. *CoRR*, abs/1609.06736, 2016. URL: <http://arxiv.org/abs/1609.06736>.
- 15 Oded Goldreich and Dana Ron. On testing expansion in bounded-degree graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 7(20), 2000.
- 16 Piotr Indyk, Reut Levi, and Ronitt Rubinfeld. Approximating and testing k-histogram distributions in sub-linear time. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 15–22, 2012. doi:10.1145/2213556.2213561.
- 17 Liam Paninski. A coincidence-based test for uniformity given very sparsely sampled discrete data. *IEEE Transactions on Information Theory*, 54(10):4750–4755, 2008.
- 18 Rocco A. Servedio. Testing by implicit learning: A brief survey. In *Property Testing – Current Research and Surveys [outgrow of a workshop at the Institute for Computer Science (ITCS) at Tsinghua University, January 2010]*, pages 197–210, 2010. doi:10.1007/978-3-642-16367-8\_11.
- 19 Gregory Valiant and Paul Valiant. An automatic inequality prover and instance optimal identity testing. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 51–60, 2014. doi:10.1109/FOCS.2014.14.
- 20 Paul Valiant. Testing symmetric properties of distributions. *SIAM J. Comput.*, 40(6):1927–1968, 2011. doi:10.1137/080734066.

# Matrix Rigidity from the Viewpoint of Parameterized Complexity

Fedor V. Fomin<sup>1</sup>, Daniel Lokshtanov<sup>2</sup>, S. M. Meesum<sup>3</sup>,  
Saket Saurabh<sup>4</sup>, and Meirav Zehavi<sup>5</sup>

- 1 University of Bergen, Bergen, Norway  
fomin@ii.uib.no
- 2 University of Bergen, Bergen, Norway  
daniello@ii.uib.no
- 3 The Institute of Mathematical Sciences, Chennai, India  
meesum@imsc.res.in
- 4 University of Bergen, Bergen, Norway; and  
The Institute of Mathematical Sciences, Chennai, India  
saket@imsc.res.in
- 5 University of Bergen, Bergen, Norway  
meirav.zehavi@ii.uib.no

---

## Abstract

The *rigidity* of a matrix  $A$  for a target rank  $r$  over a field  $\mathbb{F}$  is the minimum Hamming distance between  $A$  and a matrix of rank at most  $r$ . Rigidity is a classical concept in Computational Complexity Theory: constructions of rigid matrices are known to imply lower bounds of significant importance relating to arithmetic circuits. Yet, from the viewpoint of Parameterized Complexity, the study of central properties of matrices in general, and of the rigidity of a matrix in particular, has been neglected. In this paper, we conduct a comprehensive study of different aspects of the computation of the rigidity of *general matrices* in the framework of Parameterized Complexity. Naturally, given parameters  $r$  and  $k$ , the MATRIX RIGIDITY problem asks whether the rigidity of  $A$  for the target rank  $r$  is at most  $k$ . We show that in case  $\mathbb{F} = \mathbb{R}$  or  $\mathbb{F}$  is any finite field, this problem is fixed-parameter tractable with respect to  $k + r$ . To this end, we present a dimension reduction procedure, which may be a valuable primitive in future studies of problems of this nature. We also employ central tools in Real Algebraic Geometry, which are not well known in Parameterized Complexity, as a black box. In particular, we view the output of our dimension reduction procedure as an algebraic variety. Our main results are complemented by a  $W[1]$ -hardness result and a subexponential-time parameterized algorithm for a special case of MATRIX RIGIDITY, highlighting the different flavors of this problem.

**1998 ACM Subject Classification** G.2.2 Graph Algorithms, I.1.2 Analysis of Algorithms

**Keywords and phrases** Matrix Rigidity, Parameterized Complexity, Linear Algebra

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.32

## 1 Introduction

The *rigidity of a matrix* is a classical concept in Computational Complexity Theory, which was introduced by Grigoriev [7, 8] in 1976 and by Valiant [22] in 1977. Constructions of rigid matrices are known, for instance, to imply lower bounds of significant importance relating to arithmetic circuits. Yet, from the viewpoint of Parameterized Complexity, the study of central properties of matrices in general, and of the rigidity of a matrix in particular, has been neglected. The few papers that do consider such properties are restricted to the very



© Fedor V. Fomin, Daniel Lokshtanov, S. M. Meesum, Saket Saurabh, and Meirav Zehavi;  
licensed under Creative Commons License CC-BY

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 32; pp. 32:1–32:14

Leibniz International Proceedings in Informatics



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

special case of adjacency matrices, and therefore they are primarily studied in Graph Theory rather than Matrix Theory [16, 17]. In this paper, we conduct a comprehensive study of different aspects of the computation of the rigidity of *general matrices* in the framework of Parameterized Complexity.

Formally, given a matrix  $A$  over a field  $\mathbb{F}$ , the rigidity of  $A$ , denoted by  $\mathcal{R}_A^{\mathbb{F}}(r)$ , is defined as the minimum Hamming distance between  $A$  and a matrix of rank at most  $r$ . In other words,  $\mathcal{R}_A^{\mathbb{F}}(r)$  is the minimum number of entries in  $A$  that need to be edited in order to obtain a matrix of rank at most  $r$ . Naturally, given parameters  $r$  and  $k$ , the MATRIX RIGIDITY problem asks whether  $\mathcal{R}_A^{\mathbb{F}}(r) \leq k$ . The case when  $\mathbb{F} = \mathbb{Q}$  or the edited entries must contain integers, it is not even known whether the problem is decidable [19]. We therefore focus on the cases where  $\mathbb{F} = \mathbb{R}$  or  $\mathbb{F}$  is any finite field. Formally, we study the following forms of MATRIX RIGIDITY. Here, FF MATRIX RIGIDITY is not restricted to a specific finite field  $\mathbb{F}_p$ , but includes  $\mathbb{F}_p$  as part of the input.

REAL MATRIX RIGIDITY

**Parameter:**  $r, k$

**Input:** A matrix  $A$  with each entry an integer, and two non-negative integers  $r, k$ .

**Question:** Is  $\mathcal{R}_A^{\mathbb{R}}(r) \leq k$ ?

FF MATRIX RIGIDITY

**Parameter:**  $p, r, k$

**Input:** A finite field  $\mathbb{F}_p$  of order  $p$ , a matrix  $A$  over  $\mathbb{F}_p$ , and two non-negative integers  $r, k$ .

**Question:** Is  $\mathcal{R}_A^{\mathbb{F}_p}(r) \leq k$ ?

Valiant [22] presented the notion of the rigidity of a matrix as a means to prove lower bounds for linear algebraic circuits. He showed that the existence of an  $n \times n$  matrix  $A$  with  $\mathcal{R}_A^{\mathbb{R}}(en) \geq n^{1+\delta}$  would imply that the linear transformation defined by  $A$  cannot be computed by any arithmetic circuit having size  $\mathcal{O}(n)$  and depth  $\mathcal{O}(\log n)$  in which each gate of the circuit computes a linear combination of its inputs. Later, Razborov [18] (see [14]) established relations between lower bounds on rigidity of matrices over the reals or finite fields and strong separation results in Communication Complexity. Although many efforts have been made in this direction [6, 20, 12, 10] (this is not an exhaustive list), proofs of separation lower bounds (quadratic) for explicit families of matrices still remains elusive. For a recent survey on this topic we refer the reader to [13]. The formulation of the MATRIX RIGIDITY as stated in this paper was first considered by Mahajan and Sharma [15], and it was shown to be NP-Hard for any field by Deshpande [4]. In this paper, we study the concept of the rigidity of a matrix from a different perspective, given by the framework of Parameterized Complexity (see Section 2).

We remark that Meesum et al. [16] and Meesum and Saurabh [17] studied the following problems, which are related to MATRIX RIGIDITY but are simpler than it as they are restricted to graphs. Given a graph  $G = (V, E)$  and two non-negative integers  $r, k$ ,  $r$ -RANK VERTEX DELETION ( $r$ -RANK EDGE DELETION) asks whether one can delete at most  $k$  vertices (resp. edges) from  $G$  so that the rank of its adjacency matrix would be at most  $r$ , while  $r$ -RANK EDGE EDITING asks whether one can edit  $k$  edges in  $G$  so that the rank of its adjacency matrix would be at most  $r$ .<sup>1</sup> For undirected graphs, Meesum et al. [16] proved that these problems are NP-Hard even if  $r$  is fixed, but can be solved in time  $\mathcal{O}^*(2^{\mathcal{O}(k \log r)})$ . They also showed that  $r$ -RANK EDGE DELETION and  $r$ -RANK EDGE EDITING can be solved

<sup>1</sup> Editing an edge  $\{u, v\}$  means that if  $\{u, v\} \in E$  then  $\{u, v\}$  is deleted, and otherwise it is added.



in time  $\mathcal{O}^*(2^{\mathcal{O}(f(r)\sqrt{k}\log k)})$ . Meesum and Saurabh [17] obtained similar results for directed graphs.

**Our Contribution.** In this paper, we establish that both REAL MATRIX RIGIDITY and FF MATRIX RIGIDITY are FPT with respect to  $r+k$ . Specifically, we obtain the following results.

► **Theorem 1.** REAL MATRIX RIGIDITY can be solved in time  $\mathcal{O}^*(2^{\mathcal{O}((r+k)\cdot\log(r\cdot k))})$ .

► **Theorem 2.** FF MATRIX RIGIDITY can be solved in time  $\mathcal{O}^*(f(r, k))$  for a function  $f$  that depends only on  $r$  and  $k$ .

Observe that the dependency of the running times on the dimension of the input matrix is polynomial, and in the case of FF MATRIX RIGIDITY, the dependency of the running time on  $p$  is also polynomial. In the case of REAL MATRIX RIGIDITY, the dependency of the running time on the maximum length (in binary) of any entry in *both* input and output matrices is polynomial. In this context, recall that in case  $\mathbb{F} = \mathbb{Q}$  or the edited entries must contain integers, it is not even known whether MATRIX RIGIDITY is decidable [19]. We also show that,

► **Theorem 3** ( $\star^2$ ). FF MATRIX RIGIDITY is solvable in time  $\mathcal{O}^*(2^{\mathcal{O}(f(r,p)\sqrt{k}\log k)})$  for some function  $f$  that depends only on  $r$  and  $p$ .

Here, the dependency of the running time on  $k$  is subexponential, but the dependency of the running time on  $p$  is unsatisfactory in case  $p$  is not fixed. This algorithm adapts ideas from the papers [16, 17].

To obtain our main results, we first present a dimension reduction procedure, which we believe to be a valuable primitive in future studies of problems of this nature. Our procedure is simple to describe and given an instance of MATRIX RIGIDITY, it outputs (in polynomial time) an equivalent instance where the matrix contains at most  $\mathcal{O}((r \cdot k)^2)$  entries. Furthermore, the set of entries of the output matrix is a subset of the set of entries of the input matrix. We believe this procedure to be of interest independent of our main results as it establishes that FF MATRIX RIGIDITY admits a polynomial kernel with respect to  $r+k+p$ . The simplicity of our procedure also stems from its modularity—it handles rows and columns in separate phases. On a high-level, this procedure is defined as follows. For  $k+1$  steps, it repeatedly selects a set of maximum size consisting of rows that are linearly independent, where if the size of this set exceeds  $r+1$ , it is replaced by a subset of size exactly  $r+1$ . Each such set of rows is removed from the input matrix, and then it is inserted into the output matrix. At the end of this greedy process, rows that remain in the input matrix are simply discarded. The correctness of our procedure relies on two key insights: (i) if the input instance contains more than  $k+1$  pairwise-disjoint sets of rows that are linearly independent, and each of these sets is of size at least  $r+1$ , then the input instance is a NO-instance; (ii) by the pigeonhole principle, any row discarded from the input matrix belongs to the span of at least one set of rows that cannot be edited. Having an intermediate matrix with a small number of rows, the procedure applies the exact same process to the input that is the transpose of this intermediate matrix, thus overall obtaining a matrix with a small number of entries.

Armed with our dimension reduction procedure, we tackle REAL MATRIX RIGIDITY and FF MATRIX RIGIDITY by employing central tools in Algebraic Geometry, which are not well

<sup>2</sup> Due to space constraints, proofs marked by  $\star$  are omitted.

known in Parameterized Complexity, as a black box. For this purpose, we first recall that the rank of a matrix is at most  $r$  if and only if the determinant of all of its  $(r + 1) \times (r + 1)$  submatrices is 0. Since at this point we can assume that we have a matrix containing only  $\mathcal{O}((r \cdot k)^2)$  entries at hand, we may “guess” *which* entries should be edited. Yet, it is not clear *how* these entries should be edited. However, with the above observation in mind, we are able to proceed by viewing our current problem in terms of an algebraic variety (such a formulation was also used in the context of complexity analysis in [19]). In particular, this viewpoint gives rise to the applicability of firmly established tools that determine the feasibility of a system of polynomials [1, 9].

Our main results are complemented by a  $W[1]$ -hardness result and a subexponential-time parameterized algorithm for a special case of MATRIX RIGIDITY, which overall present the different flavors of this problem and the techniques relevant to its study. We show that both REAL MATRIX RIGIDITY and FF MATRIX RIGIDITY are  $W[1]$ -hard with respect to the parameter  $k$ . (The papers [16, 17] already imply that both of these problems are para-NP-Hard with respect to the parameter  $r$ .) Our reduction is inspired by studies in Parameterized Complexity that involve the ODD SET problem [5], and consists of four reductions, one of which builds upon the recent work of Bonnet et al. [2].

The complexity of our reduction stems from the fact that unlike previous studies of this nature, we establish the  $W[1]$ -hardness of our problem of interest over *any* finite field and over the field of reals rather than only over a specific finite field. Thus, we first need to define a special case of ODD SET, which we call PARTITIONED ODD SET, and observe that its  $W[1]$ -hardness follows from the proof of the  $W[1]$ -hardness of ODD SET that is given in [5]. The correctness of our reductions crucially relies on the implications of the properties of this special case. Our first reduction translates PARTITIONED ODD SET to a problem involving matrices rather than sets, which we call PARTITIONED ODD MATRIX. Then, to be able to discuss any finite field as well as the field of reals, we introduce new variants of PARTITIONED ODD MATRIX and the NEAREST CODEWORD problem, called  $\mathbb{F}$ -ODD MATRIX and  $\mathbb{F}$ -NEAREST CODEWORD, respectively. The application of our second reduction results in an instance of  $\mathbb{F}$ -ODD MATRIX. Then, the application of our third reduction, which builds upon [2], results in an instance of  $\mathbb{F}$ -NEAREST CODEWORD. Finally, we devise a reduction whose application results in an instance of MATRIX RIGIDITY. Here, we make explicit use of the fact that the rank of the target matrix can be large. The overall structure of the reduction may be relevant to studies of other problems where the field is not fixed.

## 2 Preliminaries

Due to space constraints, some standard notations and definitions have been omitted. The notation  $[n]$  is used to denote the set of integers  $\{1, \dots, n\}$ . Given a matrix  $A$  of dimension  $m \times n$ , the  $i$ -th row of  $A$  is denoted by  $A_i$ , and the  $j$ -th column of  $A$  is denoted by  $A^j$ . The *rank* of a matrix is the cardinality of a maximum-sized collection of linearly independent columns (or rows), and is denoted by  $\text{rank}(A)$ . We call a matrix  $\tilde{A}$  a *jumbled matrix* of  $A$  if one can perform a series of row and column exchanges on  $\tilde{A}$  to obtain the matrix  $A$ . Similarly, we call a matrix  $\tilde{A}$  a *jumbled submatrix* of  $A$  if there exists a submatrix of  $A$  which is a *jumbled matrix* of  $\tilde{A}$ . A *mixed matrix* is a matrix having either an indeterminate or a value at each entry. We will be dealing with mixed matrices where the values belong to a finite field or  $\mathbb{R}$ . We use  $I_n$  to denote the *identity matrix* of size  $n \times n$ .

Let  $x_1, \dots, x_n$  be variables. Then, a *monomial* is defined as a product  $\prod_{i=1}^n x_i^{a_i}$  for non-negative integers  $a_1, \dots, a_n$ . The degree of a variable  $x_i$  in a monomial  $\prod_{i=1}^n x_i^{a_i}$  is

defined to be the number  $a_i$ , for  $i \in [n]$ . The degree of a *monomial* is defined as the sum of degrees of each variable occurring in it. A polynomial over a field  $\mathbb{F}$  consists of a sum of monomials with coefficients from the field  $\mathbb{F}$ . The *total degree* of a polynomial is the degree of a monomial having maximum degree. Given a system of polynomial equations  $\mathcal{P} = \{P_1 = 0, P_2 = 0, \dots, P_m = 0\}$  over a field  $\mathbb{F}$ , we say that  $\mathcal{P}$  is *feasible* over  $\mathbb{F}$  if there exists an assignment of values from the field  $\mathbb{F}$  to the variables in  $\mathcal{P}$  which satisfies every polynomial contained in  $\mathcal{P}$ .

**Parameterized Complexity.** Each problem instance is associated with a parameter  $k$ , and we say that a problem is *fixed parameter tractable* (FPT) if any instance  $(I, k)$  of the problem can be solved in time  $\tau(k)|I|^{\mathcal{O}(1)}$ , where  $\tau$  is an arbitrary function of  $k$ . Throughout this paper, we use the standard notation  $\mathcal{O}^*$  to hide factors polynomial in  $|I|$ .<sup>3</sup> On the one hand, to prove that a problem is FPT, it is possible to give an explicit algorithm of the required form, called a *parameterized algorithm*, which solves it. On the other hand, to show that a problem is unlikely to be FPT, it is possible to use parameterized reductions analogous to those employed in Classical Complexity. Here, the concept of W[1]-hardness replaces the one of NP-hardness, and we need not only construct an equivalent instance in FPT time, but also ensure that the size of the parameter in the new instance depends only on the size of the parameter in the original instance. For our purposes, it is sufficient to note that if there exists such a reduction transforming a problem known to be W[1]-hard to another problem  $\Pi$ , then the problem  $\Pi$  is W[1]-hard as well.

A central notion in Parameterized Complexity is the one of *kernelization*. Formally, a parameterized problem  $\Pi$  is said to admit a *polynomial kernel* if there is a polynomial-time algorithm, called a *kernelization algorithm*, that given any instance of  $\Pi$ , outputs an equivalent instance of  $\Pi$  whose size is bounded by  $\tau(k)$ , where  $\tau$  is a function polynomial in  $k$  and independent of  $|I|$ . We say that the reduced instance is a  $p(k)$ -kernel for  $\Pi$ . Roughly speaking, a kernelization algorithm can be viewed as an efficient preprocessing procedure that satisfies a well defined restriction with respect to the size of its output. For more information about Parameterized Complexity in general and Kernelization in particular, we refer the reader to monographs such as [5, 3].

### 3 Dimension Reduction Procedure

In this section we show how to compress an input instance of MATRIX RIGIDITY to an equivalent instance in which the matrix has at most  $\mathcal{O}(r^2 \cdot k^2)$  entries. This is a crucial step in obtaining our FPT algorithms for REAL MATRIX RIGIDITY and FF MATRIX RIGIDITY. In particular, this step will imply that FF MATRIX RIGIDITY admits a polynomial kernel with respect to  $r + k + p$ .

Our algorithm is based on the following intuition. Suppose that  $A$  is a matrix of rank  $\ell$ . If we could obtain a sequence  $B_1, \dots, B_k$  of pairwise disjoint sets of columns of  $A$  where each set forms a column basis of  $A$ , then the answer to the question “can we reduce the rank of  $A$  to a number  $r < \ell$  by editing at most  $k$  entries in  $A$ ” would have been completely determined by the answer to the same question where the editing operations are restricted to the submatrix of  $A$  formed by columns in the sets  $B_1, \dots, B_k$ . The same conclusion is also true in the case where each  $B_i$  is not necessarily a basis, but simply a set of  $r + 1$  linearly independent columns. Keeping this intuition in mind, we turn to examine an approach where

<sup>3</sup> That is,  $\mathcal{O}^*(\tau(k)) = \tau(k) \cdot |I|^{\mathcal{O}(1)}$ .

we greedily select and remove (one-by-one)  $k + 1$  pairwise disjoint sets of linearly independent columns. In each iteration, we attempt to select a set whose size is exactly  $r + 1$ , where if it is not possible, we select a set of maximum size.

Now, let us move to the formal part of our arguments. Note that the relation “is a *jumbled matrix* of” as defined in Section 2 is an equivalence relation. We need the following simple observations which follow from the definition of the rank of a matrix.

► **Observation 4.** *Let  $A \in \mathbb{F}^{m \times n}$  be a matrix of rank equal to  $r$ . To make the rank of  $A$  at most  $r - 1$ , one needs to change at least one entry in  $A$ .*

► **Observation 5.** *For a matrix  $A$ , let  $\tilde{A}$  be a jumbled matrix of  $A$ . Then, the instances  $(A, r, k)$ ,  $(A^T, r, k)$ ,  $(\tilde{A}, r, k)$  and  $(\tilde{A}^T, r, k)$  are equivalent instances of MATRIX RIGIDITY.*

► **Observation 6.** *For a matrix  $A$ , let  $\tilde{A}$  be its jumbled submatrix. Then  $\text{rank}(\tilde{A}) \leq \text{rank}(A)$ .*

Using Observation 6, we have the following.

► **Observation 7.** *If  $\tilde{A}$  is a jumbled submatrix of  $A$  and  $(\tilde{A}, r, k)$  is a NO-instance of MATRIX RIGIDITY, then  $(A, r, k)$  is also a NO-instance of MATRIX RIGIDITY.*

As stated before, our procedure greedily selects a set of columns of  $A$  of appropriate dimension iteratively. A detailed description of the procedure, called COLUMN-REDUCTION, can be found in Figure 1. We will now explain the ideas necessary to understand this procedure, which is the heart of this section. The input to COLUMN-REDUCTION consists of a matrix  $A$  over any field, given along with non-negative integers  $k$  and  $r$ . It outputs a matrix  $\tilde{A}$  whose number of columns is bounded by a function of  $k$  and  $r$  such that the instances  $(A, r, k)$  and  $(\tilde{A}, r, k)$  are equivalent instances of MATRIX RIGIDITY. The computation of a column basis and linearly independent vectors are done in the field  $\mathbb{F}$  over which the matrix  $A$  is provided.

The procedure employs several variables. The variable  $i$  is used as an index variable whose initial value is 0, and it is incremented by 1 at a time. The case when the value of  $i$  exceeds  $k$  we will show that we are dealing with a NO-instance, otherwise the value depends on a particular input matrix  $A$  and is at most  $k$ . The variables  $M_0, M_1, \dots$  are submatrices of the input matrix  $A$ , satisfying the property that  $M_i$  is a submatrix of  $M_{i-1}$  with  $M_0 = A$ . In the first loop of COLUMN-REDUCTION (line 3), if the matrix  $M_i$  has rank at least  $r + 1$  then the variable  $L_i$  stores a set of  $r + 1$  linearly independent columns in the matrix  $M_i$ . Additionally,  $M_i$  can be obtained by appending the columns in  $L_i$  to the matrix  $M_{i+1}$ . The variable  $i_{\leq r}$  is set to the value of  $i$  where the rank of  $M_i$  falls below  $r + 1$ —after its initialization the value of  $i_{\leq r}$  is not changed. In the second half of the procedure, similar to the set of variables  $L_i$ , we define a set of variables  $B_i$  which store a column basis of the matrix  $M_i$  (line 6). Recall that in this half of the procedure  $i \geq i_{\leq r}$ , and therefore each matrix  $M_i$  is of rank at most  $r$ . Additionally,  $M_i$  can be obtained by appending the columns in  $B_i$  to the matrix  $M_{i+1}$ , for  $i \geq i_{\leq r}$ . Finally, the matrix  $\mathcal{L}$  is constructed using all the columns in each matrix  $L_i$ , and the matrix  $\mathcal{B}$  is constructed using all the columns in each matrix  $B_i$  for appropriate values of  $i$ . By Observation 4, we have to edit at least  $i_{\leq r}$  entries of  $\mathcal{L}$  to make its rank at most  $r$ .

In the procedure COLUMN-REDUCTION, a YES-instance of appropriate size can be obtained by taking the matrix  $Z = [0]$  (of rank 0), which contains 0 as its only entry. Clearly,  $(Z, r, k)$  is a YES-instance of MATRIX RIGIDITY irrespective of the values of  $r$  and  $k$ . On the other hand, the instance  $(I_{r+k+1}, r, k)$  is a NO-instance of MATRIX RIGIDITY. Therefore, the matrix  $I_{r+k+1}$  can be used in place of a NO-instance of appropriate size. We need  $Z$  and

**Algorithm:** COLUMN-REDUCTION

*Input:* A matrix  $A$  over some field  $\mathbb{F}$ , and two non-negative integers  $r, k$ .

*Output:* A matrix having  $\mathcal{O}(r \cdot k)$  columns.

1. If  $\text{rank}(A) \leq r$  then return a YES-instance of appropriate size and exit.
2. Initialize  $M_0 = A$  and  $i = 0$ .
3. While  $\text{rank}(M_i) \geq r + 1$ :
  - a. Let  $L_i$  be a set of columns of  $M_i$  which is linearly independent in  $\mathbb{F}$  and whose size is  $r + 1$ .
  - b. Let  $M_{i+1}$  be the matrix obtained by deleting the columns in  $L_i$  from  $M_i$ .
  - c. Increment  $i$  by 1.
4. If  $i > k$  then return a NO-instance of appropriate size and exit.  
 // The matrix  $A$  has more than  $k$  pairwise-disjoint blocks of the form  $L_j$  for  $j \leq i$ , each having  $r + 1$  linearly independent columns. By Observation 4, each block  $L_i$  requires at least 1 edit, hence, by Observation 7,  $(A, r, k)$  is a NO-instance of MATRIX RIGIDITY.
5. Let  $i_{\leq r} = i$  store the index where the rank of  $M_i$  falls below  $r + 1$ .
6. While  $i \leq k$ :
  - a. Let  $B_i$  be a column basis of  $M_i$ .
  - b. Obtain  $M_{i+1}$  by deleting the columns in  $B_i$  from  $M_i$ .
  - c. If  $M_{i+1}$  is empty (in other words,  $B_i = M_i$ ) then return  $A$ .
  - d. Increment  $i$  by 1.
7. Let  $\mathcal{L}$  be a matrix formed by the columns in each  $L_i$  for  $i \in \{0, \dots, i_{\leq r} - 1\}$ .
8. Let  $\mathcal{B}$  be a matrix formed by the columns in each  $B_i$  for  $i \in \{i_{\leq r}, \dots, k\}$ .
9. Return the matrix formed by the columns in  $\mathcal{L} \cup \mathcal{B}$ .  
 //Note that  $M_{k+1}$  is non-empty if output occurs here.

■ **Figure 1** The column reduction procedure.

$I_{r+k+1}$  to satisfy the constraint that a kernel is an instance of the same problem as the input instance (even though, if the output is given by either line 1 or 4, we have actually solved the input instance  $(A, r, k)$  of MATRIX RIGIDITY in polynomial time). Using the procedure COLUMN-REDUCTION, it is straightforward to reduce the number of rows too. The details of this procedure are given in Figure 2.

► **Lemma 8.** *Let  $A$  be a matrix over some field  $\mathbb{F}$ , and let  $r$  and  $k$  be two non-negative integers. Given an instance  $(A, r, k)$ , the procedure MATRIX-REDUCTION runs in time polynomial in input size and returns a matrix  $\tilde{A}$  satisfying the following properties:*

1.  $\tilde{A}$  has at most  $\mathcal{O}(r^2 \cdot k^2)$  entries.
2. If the output is produced by lines 6c and 9 of COLUMN-REDUCTION (when called by MATRIX-REDUCTION), then  $\tilde{A}$  is a jumbled submatrix of  $A$ .
3.  $(A, r, k)$  is a YES-instance of MATRIX RIGIDITY if and only if  $(\tilde{A}, r, k)$  is a YES-instance.

**Proof.** The steps of procedure COLUMN-REDUCTION are all computable in polynomial time, and therefore MATRIX-REDUCTION runs in polynomial time. We now prove the desired properties one by one. Let the matrix  $\tilde{N}$  denote the output of COLUMN-REDUCTION on the input instance  $(N, r, k)$ .

**Proof of 1.** We first bound the size of the output of COLUMN-REDUCTION. The output of this procedure can occur at lines 1, 4, 6c and 9. If the output happens at line 1, it has 1 column

**Algorithm:** MATRIX-REDUCTION

*Input:* A matrix  $A$  over some field  $\mathbb{F}$ , and two non-negative integers  $r, k$ .

*Output:* A matrix having  $\mathcal{O}(r \cdot k) \times \mathcal{O}(r \cdot k)$  entries.

1. Let  $C_A = \text{COLUMN-REDUCTION}(A, r, k)$ .
2. Let  $R_A = \text{COLUMN-REDUCTION}(C_A^T, r, k)$ .
3. Return  $R_A^T$ .

■ **Figure 2** The dimension reduction procedure.

by construction. Similarly, if the output happens at line 4, it has  $r + k + 1 \leq (r + 1) \cdot (k + 1)$  columns by construction. If the output occurs at line 6c or line 9, then the number of columns in  $\tilde{N}$  is at most  $(k + 1) \cdot (r + 1)$  as it is constructed using columns of at most  $i \leq k$  matrices,  $L_0, \dots, L_{i \leq r-1}, B_{i \leq r}, \dots, B_i$ , each having at most  $r + 1$  columns.

The procedure MATRIX-REDUCTION first obtains a matrix  $C_A$  with the aforementioned number of columns by running COLUMN-REDUCTION. Then, it runs COLUMN-REDUCTION again on the transpose of  $C_A$  to get its rows bounded. Thus, the dimensions of the output matrix are as claimed. ◀

**Proof of 2.** The relation “is a *jumbled submatrix* of” is a transitive relation, therefore it suffices to show that the procedure COLUMN-REDUCTION outputs a *jumbled submatrix* of  $A$ . If the output happens at lines 6c and 9, then the columns in the output matrix are a subset of the columns in the input matrix. Therefore, in the first line of procedure MATRIX-REDUCTION  $C_A$  is a *jumbled submatrix* of  $A$ . Similarly,  $R_A$  is a *jumbled submatrix* of  $C_A^T$ . Finally note that for matrices  $X$  and  $Y$ ,  $X$  is a *jumbled submatrix* of  $Y$  if and only if  $X^T$  is a *jumbled submatrix* of  $Y^T$ . Hence, the output matrix  $R_A^T$  is a *jumbled submatrix* of  $A$ . ◀

**Proof of 3:** We first show that the procedure COLUMN-REDUCTION produces an equivalent instance of MATRIX RIGIDITY. In the forward direction, suppose that  $(N, r, k)$  is a YES-instance of MATRIX RIGIDITY. If the output occurs at line 1, it is a YES-instance by construction. The output cannot occur at line 4 as  $(N, r, k)$  is a YES-instance. At lines 6c and 9, by property 2, COLUMN-REDUCTION outputs a *jumbled submatrix*  $\tilde{N}$  of the input matrix  $N$ . Let  $S$  denote a solution of instance  $(N, r, k)$  of MATRIX RIGIDITY. The set  $S$  consists of replaced entries and their indices. We denote the edited matrix obtained from  $N$  by  $N_S$ . We construct  $\tilde{S}$ , a solution of  $(\tilde{N}, r, k)$ , by remembering the new positions of columns and rows of  $N$  in the matrix  $\tilde{N}$ , and then performing the same permutations on the indices of entries in  $S$ . As  $\tilde{N}_{\tilde{S}}$  is a *jumbled submatrix* of  $N_S$ , by Observation 6,  $\text{rank}(\tilde{N}_{\tilde{S}}) \leq \text{rank}(N_S) \leq r$ , hence  $(\tilde{N}, r, k)$  is a YES-instance of MATRIX RIGIDITY. ◀

In the backward direction, suppose  $(\tilde{N}, r, k)$  is a YES-instance of MATRIX RIGIDITY. If the output of  $\tilde{N}$  occurs at lines 1 or 4, then we actually know the solution to the instance  $(N, r, k)$  of MATRIX RIGIDITY (as explained in the pseudocode). If the output occurs at line 6c, then the output  $\tilde{N}$  of COLUMN-REDUCTION is a *jumbled matrix* of  $N$  and the result holds by Observation 5. Now we are left with the case when the output occurs at line 9. Let  $\tilde{S}$  be any solution to the instance  $(\tilde{N}, r, k)$  of MATRIX RIGIDITY. The matrix edited using a solution  $\tilde{S}$  is denoted by  $\tilde{N}_{\tilde{S}}$ . Notice that the matrix  $\tilde{N}$  consists of two submatrices  $\mathcal{L}$  and  $\mathcal{B}$ . As  $\mathcal{L}$  consists of  $i \leq r$  blocks having rank  $r + 1$ , by Observation 4, we need to edit at least  $i \leq r$  entries in  $\mathcal{L}$ . So, we can afford to make at most  $k - i \leq r$  edits in the matrix  $\mathcal{B}$ . As  $\mathcal{B}$  consists of  $k + 1 - i \leq r$  blocks, by pigeonhole principle there exists at least one block in  $\mathcal{B}$ , say  $B_t$ ,

which is not subject to any edit by the solution  $\tilde{S}$ . The block  $B_t$  has rank at most  $r$  and spans the matrix  $M_{k+1}$  (refer to pseudocode), by construction. Construct the matrix  $N'_S$  by concatenating the columns of  $\tilde{N}_{\tilde{S}}$  and  $M_{k+1}$ ; it has rank at most  $r$  as the columns of the matrix  $M_{k+1}$  are in the span of the columns of  $\tilde{N}_{\tilde{S}}$ . Similarly, construct another matrix  $N'$  by concatenating the columns of  $\tilde{N}$  and  $M_{k+1}$ . As observed in the first half of this proof, we can easily construct a solution  $S$  for the matrix  $N$  using  $N'_S$ , as  $N'$  is a *jumbled matrix* of  $N$ . Thus,  $\text{rank}(N_S) = \text{rank}(N'_S) = \text{rank}(\tilde{N}_{\tilde{S}}) \leq r$ , proving that the instance  $(N, r, k)$  is a YES-instance of MATRIX RIGIDITY.

To complete the proof, observe that in the procedure MATRIX-REDUCTION, the instances  $(A, r, k)$  and  $(C_A, r, k)$  are equivalent by the argument above. By Observation 5,  $(C_A, r, k)$  and  $(C_A^T, r, k)$  are equivalent. As  $R_A$  is the output of COLUMN-REDUCTION,  $(R_A, r, k)$  is equivalent to  $(C_A, r, k)$ . Finally, by Observation 5, again  $(R_A, r, k)$  and  $(R_A^T, r, k)$  are equivalent. ◀

If the matrix  $A$  is over a fixed finite field  $\mathbb{F}$ , we obtain a kernel as well.

► **Theorem 9.** *Given an instance  $(A, r, k)$  of FF MATRIX RIGIDITY over the field  $\mathbb{F}_p$ , the procedure MATRIX-REDUCTION outputs an  $\mathcal{O}(r^2 \cdot k^2 \cdot \log p)$ -kernel.*

**Proof.** The number of entries in the output matrix of MATRIX-REDUCTION is bounded by  $\mathcal{O}(r^2 \cdot k^2)$ , and the bit length of each entry is at most  $\lceil \log_2 p \rceil$ . ◀

In case the field  $\mathbb{F}$  is infinite—for example, if  $\mathbb{F}$  is either  $\mathbb{Q}$  or  $\mathbb{R}$ —the procedure is not guaranteed to produce a kernel as the bit lengths of matrix entries may not be bounded by a function of  $r$  and  $k$ .

#### 4 Fixed-Parameter Tractability with Respect to $k + r$

This section describes an algorithm for MATRIX RIGIDITY. The formulation it presents was also used in the context of complexity analysis in [19].

Using Lemma 8, we can reduce any instance  $(A, r, k)$  to an equivalent instance  $(A', r, k)$  such that the matrix  $A'$  is a *jumbled submatrix* of  $A$  and the number of entries in  $A'$  is  $\mathcal{O}(r^2 \cdot k^2)$ . Once we have such a matrix  $A'$ , it is useful to examine an alternative definition of the rank of a matrix, which is given in terms of the determinant of its square submatrices. Specifically, we will rely on the following lemma.

► **Lemma 10** (Chapter 7, [21]). *A matrix  $A$  over  $\mathbb{R}$  has rank at most  $r$  if and only if all the  $(r + 1) \times (r + 1)$  submatrices of  $A$  have determinant 0.*

The correctness of our algorithm MATRIG-ALG for MATRIX RIGIDITY, which is described in Figure 3, follows in a straightforward fashion using Lemma 10. This algorithm for MATRIX RIGIDITY crucially relies on a procedure which can decide the feasibility of a system of polynomials over a given field. This procedure shall be the object of discussion in the rest of the section.

Observe that each polynomial in  $\mathcal{P}$ , as defined in the algorithm MATRIG-ALG, has at most  $k$  unknowns and its *total degree* is at most  $k$ . The size of  $\mathcal{P}$  is of order  $(r \cdot k)^{\mathcal{O}(r)}$ . In the case where the underlying field is  $\mathbb{R}$ , suppose the longest length entry has bit length  $L$ . The coefficients of polynomials in  $\mathcal{P}$  are obtained by computing the determinant of matrices which have size at most  $r \times r$ . By *Hadamard's inequality* [11], for a matrix  $M$  of size  $r \times r$  the  $\det(M) \leq \prod_{i \in [r]} \|M_i\|_2$ . If the bit length of entries in  $M$  is at most  $L$  then the coefficients of polynomials in  $\mathcal{P}$  can be shown to be bounded by  $L' = r \cdot L + \mathcal{O}(r \cdot \log r)$ .

**Algorithm:** MATRIG-ALG

*Input:* A matrix  $A$  over a field  $\mathbb{F}$ , and two non-negative numbers  $r, k$ .

*Output:* Can one edit at most  $k$  entries of  $A$  to obtain a matrix of rank at most  $r$ ?

1. Let  $A' = \text{MATRIX-REDUCTION}(A, r, k)$ .
2. For each set  $E$  of  $k$  entries in  $A'$ :
  - a. Replace each entry of  $A'$  indexed by an element in  $E$  by a distinct indeterminate to obtain a *mixed matrix*  $A'_E$ .
  - b. Let  $\mathcal{P}$  be the set of equations obtained by setting the determinant of each  $(r + 1) \times (r + 1)$  submatrix of  $A'_E$  to 0.
  - c. If  $\mathcal{P}$  is feasible over  $\mathbb{F}$  then return YES and exit.
3. Return NO and exit.

■ **Figure 3** Description of the algorithm for MATRIX RIGIDITY.

We use the following proposition, given in [1] (Proposition 13.19), to check the feasibility of the system of polynomials  $\mathcal{P}$  when it is defined over  $\mathbb{R}$ .

► **Proposition 11.** *Given a set  $\mathcal{P}$  of  $\ell$  polynomials of degree  $d$  in  $k$  variables with coefficients in  $C$ , we can decide with complexity  $\ell \cdot d^{\mathcal{O}(k)}$  in  $D$  (where  $D$  is the ring generated by the real and imaginary parts of the coefficients of the polynomials in  $\mathcal{P}$ ) whether  $\text{Zer}(\mathcal{P}, C^k)$  is empty. Moreover, if  $D = \mathbb{Z}$  and the bit-sizes of the coefficients of the polynomials are bounded by  $\tau$ , then bit-sizes of the integers appearing in the intermediate computations and the output are bounded by  $(\tau + \log \ell) \cdot d^{\mathcal{O}(k)}$ .*

Applying the proposition above on the system of equations  $\mathcal{P}$ , we get the following.

► **Theorem 12.** *Suppose we are given a matrix  $A$  over  $\mathbb{R}$  such that the bit length of each of its entries is bounded by  $L$ , and let  $r$  and  $k$  be two non-negative integers. Then, the instance  $(A, r, k)$  of REAL MATRIX RIGIDITY can be solved in time  $\mathcal{O}^*(2^{\mathcal{O}((r+k) \cdot \log(r \cdot k))})$ . Here, the bit lengths of integers appearing in intermediate computations and the output are bounded by  $\mathcal{O}(r \cdot (\log(k \cdot r) + L)) \cdot 2^{\mathcal{O}(k \cdot \log k)}$ .*

**Proof.** The algorithm MATRIG-ALG generates  $\mathcal{O}((r \cdot k)^{2k})$  many systems of equations. Each system of equations has  $\ell = (r \cdot k)^{\mathcal{O}(r)}$  many equations, where the degree  $d = k$  and there are  $k$  variables. Using Proposition 11, we get the required running time and the bit lengths.

Notice that a system of equations  $\mathcal{P}$  is feasible if and only if the chosen entries of the matrix can be edited to reduce the rank. Since we exhaustively try all possible entries that can be edited, the correctness of MATRIG-ALG follows. ◀

In the case where the underlying field  $\mathbb{F}_p$  is finite, the coefficients of the polynomials are elements of  $\mathbb{F}_p$  and hence have bounded bit lengths. The feasibility of  $\mathcal{P}$  over a finite field can be decided using the following known algorithm which also gives us an algorithm for FF MATRIX RIGIDITY.

► **Proposition 13** ([9]). *There is a deterministic algorithm which, given an input consisting of a finite field  $\mathbb{F}_p$  and system of polynomials  $f_1, \dots, f_\ell \in \mathbb{F}_p[x_1, \dots, x_k]$  of total degree bounded by  $d$ , decides the feasibility of the system in time  $d^{k \cdot \mathcal{O}(k)} \cdot (\ell \cdot \log p)^{\mathcal{O}(1)}$ .*

Similar to the proof of Theorem 12, we obtain the following.



► **Theorem 14.** *The problem FF MATRIX RIGIDITY, where the input matrix  $A$  is an  $m \times n$  matrix over a field  $\mathbb{F}_p$ , can be solved in time  $f(r, k)(\log p + m + n)^{\mathcal{O}(1)}$  for some function  $f$ .*

This algorithm for FF MATRIX RIGIDITY has the advantage that it runs in time which is *polynomial in the logarithm of the order of the field*, even though the dependence on  $k$  is exponential.

## 5 W[1]-Hardness with Respect to $k$

In this section, we first reduce (in two steps) a special case of ODD SET to a problem that has a formulation easier to use in our context. The latter problem is reduced to a variant of NEAREST CODEWORD, which, in its turn, is reduced to REAL MATRIX RIGIDITY and FF MATRIX RIGIDITY.

► **Theorem 15.** *REAL MATRIX RIGIDITY and FF MATRIX RIGIDITY for any choice of a finite field  $\mathbb{F}_p$  are W[1]-hard with respect to  $k$ .*

**Proof.** Let  $\mathbb{F}$  denote the field, which is either  $\mathbb{R}$  or some finite field  $\mathbb{F}_p$ , over which we define MATRIX RIGIDITY. First, we observe that the reduction from MULTICOLORED CLIQUE given in the book [3] (Theorem 13.31) to show that ODD SET is W[1]-hard actually shows that the following special case of ODD SET is W[1]-hard. That is, the constructed instances have the form specified in the special case.

PARTITIONED ODD SET

**Parameter:**  $k$

**Input:** A family  $\mathcal{F}$  of sets over a universe  $U$ , a non-negative integer  $k$ , a partition  $(U_1, \dots, U_k)$  of  $U$  such that for every  $i \in [k]$ ,  $U_i \in \mathcal{F}$ , and for every  $F \in \mathcal{F}$ , there exist  $i, j \in [k]$  for which  $F \subseteq U_i \cup U_j$ .

**Question:** Is there a subset  $S \subseteq U$  of size at most  $k$  such that the intersection of  $S$  with every set in  $\mathcal{F}$  has size 1?

The arguments below will crucially rely on the fact that we restrict ourselves to this special case. Given a vector  $v$ , we let  $\bar{I}_0(v)$  denote the indices of the entries of  $v$  that do not contain 0. Now, we reformulate PARTITIONED ODD SET in the language of matrices as follows.

PARTITIONED ODD MATRIX

**Parameter:**  $k$

**Input:** A  $t \times r$  binary matrix  $L$  over  $\mathbb{R}$ , a non-negative integer  $k$ , a partition  $(U_1, \dots, U_k)$  of  $[r]$  such that for every  $i \in [k]$ , there exists  $j \in [t]$  for which  $U_i = \bar{I}_0(L_j)$ , and for every  $i \in [t]$ , there exist  $j, \ell \in [k]$  for which  $\bar{I}_0(L_i) \subseteq U_j \cup U_\ell$ .

**Question:** Is there an  $r$ -dimensional binary vector  $x$  such that  $|\bar{I}_0(x)| \leq k$  and  $Lx = 1$ ?

Given an instance  $(\mathcal{F}, U, (U_1, \dots, U_k), k)$  of PARTITIONED ODD SET, it is straightforward to obtain (in polynomial time) an equivalent instance  $(L_{t \times r}, (U'_1, \dots, U'_k), k')$  of PARTITIONED ODD MATRIX as follows. First, we let  $t = |\mathcal{F}|$  and  $r = |U|$ . We assume w.l.o.g. that  $U = [r]$ . Now, we associate a row  $L_i$  with each set  $F \in \mathcal{F}$  by letting  $L_i$  contain 1 at each entry whose index belongs to  $F$  and 0 at each of the remaining entries. That is,  $\bar{I}_0(L_i) = F$ . Finally, we let  $(U'_1, \dots, U'_k) = (U_1, \dots, U_k)$  and  $k' = k$ . It is easy to see that  $S \subseteq U$  is a solution to  $(\mathcal{F}, U, (U_1, \dots, U_k), k)$  if and only if the binary vector  $x_{r \times 1}$  such that  $\bar{I}_0(x) = S$  is a solution to  $(L_{t \times r}, (U'_1, \dots, U'_k), k)$ , and therefore the instances are equivalent.

We now incorporate the input field  $\mathbb{F}$ .

**ℱ-ODD MATRIX****Parameter:**  $k$ **Input:** A  $t \times r$  binary matrix  $L$  over  $\mathbb{F}$  and a non-negative integer  $k$ .**Question:** Is there an  $r$ -dimensional vector  $x$  over  $\mathbb{F}$  such that  $|\bar{I}_0(x)| \leq k$  and  $Lx = 1$ ?

We reduce PARTITIONED ODD MATRIX to  $\mathbb{F}$ -ODD MATRIX as follows. Given an instance  $(L_{t \times r}, (U_1, \dots, U_k), k)$  of PARTITIONED ODD MATRIX, we simply output  $(L_{t \times r}, k)$  as the equivalent instance of  $\mathbb{F}$ -ODD MATRIX. In one direction, let  $x$  be a solution to  $(L_{t \times r}, (U_1, \dots, U_k), k)$ . Recall that  $L$  is a binary matrix. Thus, since  $x$  is a binary vector satisfying  $Lx = 1$  over  $\mathbb{R}$ , it must also satisfy  $Lx = 1$  over  $\mathbb{F}$ . Since  $|\bar{I}_0(x)| \leq k$ , we get that  $x$  is a solution to  $(L_{t \times r}, k)$ . In the second direction, let  $x$  be a solution to  $(L_{t \times r}, k)$ . Since  $|\bar{I}_0(x)| \leq k$  and for every  $s \in [k]$ , there exists  $s' \in [t]$  for which  $U_s = \bar{I}_0(L_{s'})$ , it must hold that for every  $s \in [k]$ ,  $|\bar{I}_0(x) \cap U_{s'}| = 1$ . Thus, since for every  $s \in [k]$ ,  $L_{s'}x = 1$  over  $\mathbb{F}$ , it holds that  $x$  is a binary vector. It remains to show that  $Lx = 1$  over  $\mathbb{R}$ . Consider some index  $i \in [t]$ . Then, there exist  $j, \ell \in [k]$  such that  $\bar{I}_0(L_i) \subseteq U_j \cup U_\ell$ . Therefore, since for every  $s \in [k]$ ,  $|\bar{I}_0(x) \cap U_s| = 1$ , and since both  $L$  and  $x$  are binary, it is only possible that  $L_i x = 1$  over  $\mathbb{F}$  if  $|\bar{I}_0(x) \cap \bar{I}_0(L_i)| = 1$ , which allows us to conclude that  $L_i x = 1$  also over  $\mathbb{R}$ .

In what follows, calculations are performed over  $\mathbb{F}$ . Next, we reduce  $\mathbb{F}$ -ODD MATRIX to the following variant of the NEAREST CODEWORD problem. This specific reduction is inspired by a reduction from NEAREST CODEWORD to ODD SET of Bonnet et al. [2].

**ℱ-NEAREST CODEWORD****Parameter:**  $k$ **Input:** An  $m \times n$  matrix  $M$ , an  $m$ -dimensional vector  $b$  over  $\mathbb{F}$ , and a non-negative integer  $k$ .**Question:** Is there an  $n$ -dimensional vector  $y$  over  $\mathbb{F}$  such that the Hamming distance between  $My$  and  $b$  is at most  $k$ ?

Given an instance  $(L_{t \times r}, k)$  of  $\mathbb{F}$ -ODD MATRIX, we construct an instance  $(M_{m \times n}, b, k')$  of  $\mathbb{F}$ -NEAREST CODEWORD as follows. First, let  $k' = k$ . Now, let  $M$  be an  $m \times n$  matrix, where  $m = r$  and  $n = r - \text{rank}(L)$ , such that the rows of  $L$  form a basis for the subspace orthogonal to the column space of  $M$ . Then, an  $r$ -dimensional vector  $v$  over  $\mathbb{F}$  satisfies  $Lv = 0$  if and only if  $v$  belongs to the column space of  $M$  (i.e., there is an  $n$ -dimensional vector  $y$  over  $\mathbb{F}$  such that  $My = v$ ). Finally, let  $b$  be an  $r$ -dimensional vector such that  $Lb = -1$ . If no such vector exists, then there is no  $r$ -dimensional vector over  $\mathbb{F}$  such that  $Lv = 1$ , which in particular implies that  $(L_{t \times r}, k)$  is a NO-instance, and thus we can return a trivial NO-instance of  $\mathbb{F}$ -NEAREST CODEWORD. Therefore, next assume that  $b$  exists. To prove that the reduction is correct, first let  $x$  be a solution to  $(L_{t \times r}, k)$ . Then,  $Lx = 1$ , and since  $Lb = -1$ , we have that  $L(x + b) = Lx + Lb = Lx - 1 = 0$ . Therefore, by the choice of  $M$ , there exists an  $n$ -dimensional vector  $y$  over  $\mathbb{F}$  such that  $My = (x + b)$ . Since  $|\bar{I}_0(x)| \leq k$ , we have that the Hamming distance between  $My$  and  $b$  is at most  $k$ , which implies that  $y$  is a solution to  $(M_{m \times n}, b, k')$ . In the other direction, let  $y$  be a solution to  $(M_{m \times n}, b, k')$ . Then, since the Hamming distance between  $My$  and  $b$  is at most  $k$ , there exists an  $m$ -dimensional vector  $x$  such that  $|\bar{I}_0(x)| \leq k$  and  $My = x + b$ . Therefore, by the choice of  $M$ ,  $L(x + b) = 0$ . Since  $Lb = -1$ , we get that  $Lx = 1$ , which implies that  $x$  is a solution to  $(L_{t \times r}, k)$ .

Finally, we reduce  $\mathbb{F}$ -NEAREST CODEWORD to MATRIX RIGIDITY over  $\mathbb{F}$ . For this purpose, let  $(M_{m \times n}, b, k)$  be an instance of  $\mathbb{F}$ -NEAREST CODEWORD. We assume w.l.o.g. that the columns of  $M$  are linearly independent. We construct an equivalent instance  $(A_{\hat{a} \times \hat{b}}, r, \hat{k})$  of MATRIX RIGIDITY over  $\mathbb{F}$  as follows. First, let  $\hat{k} = k$ . Now, let  $\hat{a} = m$ ,  $r = n$  and  $\hat{b} = (k + 1)n + 1$ . For each  $i \in [k + 1]$  and  $j \in [n]$ , we define  $A^{(k+1)(i-1)+j} = M^j$ . Finally, we

define  $A^{(k+1)m+1} = b$ . On the one hand, let  $y$  be a solution to  $(M_{m \times n}, b, k)$ . Then, there are at most  $k$  entries that should be changed in  $b$  to obtain an  $m$ -dimensional vector  $b'$  over  $\mathbb{F}$  such that  $My = b'$ . In the matrix  $A$ , replace the last column  $b$  by  $b'$ . Denote the resulting matrix by  $A'$ . Then, the last column of  $A'$  is a linear combination of its other columns (by the construction of  $A$  and since  $My = b'$ ), and among the other columns of  $A'$ , there are only  $m$  distinct columns. Therefore,  $\text{rank}(A') = n$ , which implies that  $(A_{\widehat{a \times b}}, r, \widehat{k})$  is a YES-instance. In the other direction, suppose that  $(A_{\widehat{a \times b}}, r, \widehat{k})$  is a YES-instance. Then, it is possible to change at most  $k$  entries in  $A$  and obtain a matrix  $A'$  such that  $\text{rank}(A') = n$ . Since besides the last column of  $A$ , each column of  $A$  is repeated  $k+1$  times (i.e., more times than the number of changes), and there are  $n$  such distinct rows, it must be that the last column of  $A'$  is a linear combination of the distinct columns of  $A$  excluding the last column of  $A$ . By the construction of  $A$ , we get that there exists an  $n$ -dimensional vector  $y$  over  $\mathbb{F}$  such that  $My = b'$ , where  $b'$  is the last column of  $A'$ . Since the Hamming distance between  $b$  and  $b'$  is at most  $k$ , we have that  $y$  is a solution to  $(M_{m \times n}, b, k)$ . ◀

---

## References

- 1 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
- 2 E. Bonnet, L. Egri, and D. Marx. Fixed-parameter approximability of boolean MinCSPs. In *ESA*, pages 18:1–18:18, 2016.
- 3 M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshantov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms*. Springer, 2015.
- 4 Amit Jayant Deshpande. *Sampling-based algorithms for dimension reduction*. PhD thesis, Massachusetts Institute of Technology, 2007.
- 5 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
- 6 Joel Friedman. A note on matrix rigidity. *Combinatorica*, 13(2):235–239, 1993.
- 7 D. Grigoriev. Using the notions of separability and independence for proving the lower bounds on the circuit complexity (in russian). *Notes of the Leningrad branch of the Steklov Mathematical Institute, Nauka*, 1976.
- 8 D. Grigoriev. Using the notions of separability and independence for proving the lower bounds on the circuit complexity. *Journal of Soviet Math.*, 14(5):1450–1456, 1980.
- 9 Neeraj Kayal. Solvability of a system of bivariate polynomial equations over a finite field. In *ICALP*, pages 551–562, 2005.
- 10 Abhinav Kumar, Satyanarayana V. Lokam, Vijay M. Patankar, and Jayalal M. N. Sarma. Using elimination theory to construct rigid matrices. *Computational Complexity*, 23(4):531–563, 2013.
- 11 Kenneth Lange. Hadamard’s determinant inequality. *The American Mathematical Monthly*, 121(3):258–259, 2014.
- 12 Satyanarayana V. Lokam. On the rigidity of Vandermonde matrices. *Theoretical Computer Science*, 237(1–2):477–483, 2000.
- 13 Satyanarayana V. Lokam. Complexity lower bounds using linear algebra. *Found. Trends Theor. Comput. Sci.*, 4:1–155, January 2009.
- 14 S. V. Lokam. Spectral methods for matrix rigidity with applications to size-depth tradeoffs and communication complexity. In *FOCS*, pages 6–15, 1995.
- 15 Meena Mahajan and Jayalal M. N. Sarma. On the complexity of matrix rank and rigidity. In *CSR*, pages 269–280, 2007.

## 32:14 Matrix Rigidity from the Viewpoint of Parameterized Complexity

- 16 S. M. Meesum, Pranabendu Misra, and Saket Saurabh. Reducing rank of the adjacency matrix by graph modification. In *COCOON*, pages 361–373, 2015.
- 17 S. M. Meesum and S. Saurabh. Rank reduction of directed graphs by vertex and edge deletions. In *LATIN*, pages 619–633, 2016.
- 18 A. A. Razborov. On rigid matrices. *Manuscript in russian*, 1989.
- 19 Jayalal M. N. Sarma. *Complexity Theoretic Aspects of Rank, Rigidity and Circuit Evaluation*. PhD thesis, The Institute of Mathematical Sciences, CIT Campus, Taramani, Chennai, 2009.
- 20 M. A. Shokrollahi, D. Spielman, and V. Stemann. A remark on matrix rigidity. *Information Processing Letters*, 64(6):283–285, 1997.
- 21 L. E. Sigler. *Algebra*. Undergraduate Texts in Mathematics. Springer-Verlag, 1976.
- 22 L. G. Valiant. Graph-theoretic arguments in low-level complexity. In *MFCS*, pages 162–176, 1977.

# Deterministic Regular Expressions with Back-References

Dominik D. Freydenberger<sup>\*1</sup> and Markus L. Schmid<sup>2</sup>

1 University of Bayreuth, Bayreuth, Germany  
ddfy@ddfy.de

2 University of Trier, Trier, Germany  
MSchmid@uni-trier.de

---

## Abstract

Most modern libraries for regular expression matching allow back-references (i. e., repetition operators) that substantially increase expressive power, but also lead to intractability. In order to find a better balance between expressiveness and tractability, we combine these with the notion of determinism for regular expressions used in XML DTDs and XML Schema. This includes the definition of a suitable automaton model, and a generalization of the Glushkov construction.

**1998 ACM Subject Classification** F.1.1 Models of Computation, F.4.3 Formal Languages

**Keywords and phrases** Deterministic Regular Expression, Regex, Glushkov Automaton

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.33

## 1 Introduction

Regular expressions were introduced in 1956 by Kleene [26] and quickly found wide use in both theoretical and applied computer science. While the theoretical interpretation of regular expressions remains mostly unchanged (as expressions that describe exactly the class of regular languages), modern applications use variants that vary greatly in expressive power and algorithmic properties. This paper tries to find common ground between two of these variants with opposing approaches to the balance between expressive power and tractability.

The first variant that we consider are *regex*, regular expressions that are extended with a *back-reference operator*. This operator is used in almost all modern programming languages (like e. g. Java, PERL, and .NET). For example, the regex  $\langle x: (\mathbf{a} \vee \mathbf{b})^* \rangle \cdot \&x$  defines  $\{ww \mid w \in \{\mathbf{a}, \mathbf{b}\}^*\}$ , as  $(\mathbf{a} \vee \mathbf{b})^*$  can create a  $w \in \{\mathbf{a}, \mathbf{b}\}^*$ , which is then stored in the variable  $x$  and repeated with the reference  $\&x$ . Hence, back-references allow to define non-regular languages; but with the side effect that the membership problem is NP-complete (cf. Aho [2]).

The other variant, *deterministic regular expressions* (also known as *1-unambiguous regular expressions*), uses an opposite approach, and achieves a more efficient membership problem than regular expressions by defining only a strict subclass of the regular languages.

Intuitively, a regular expression is deterministic if, when matching a word from left to right with no lookahead, it is always clear where in the expression the next symbol must be matched. This property has a characterization via the *Glushkov construction* that converts every regular expression  $\alpha$  into a (potentially non-deterministic) finite automaton  $\mathcal{M}(\alpha)$ , by treating each terminal position in  $\alpha$  as a state. Then  $\alpha$  is deterministic if  $\mathcal{M}(\alpha)$  is deterministic. As a consequence, the membership problem for deterministic regular expressions can be solved

---

\* Dominik D. Freydenberger was supported by DFG grant FR 3551/1-1.



more efficiently than for regular expressions in general (more details can be found in [24]). Hence, in spite of their limited expressive power, deterministic regular expressions are used in actual applications: Originally defined for the ISO standard for SGML (see Brüggemann-Klein and Wood [9]), they are a central part of the W3C recommendations on XML DTDs [7] and XML Schema [22] (see Murata et al. [32]).

The goal of this paper is finding common ground between these two variants, by introducing *deterministic regex* and an appropriate automaton model, the *deterministic memory automata with trap-state* (DTMFA). To elaborate: We first introduce a new automaton model for regex, the memory automata with trap-state (TMFA). While the TMFA is based on the MFA that was proposed by Schmid [35], its deterministic variant, the DTMFA, is better suited for complementation than the deterministic MFA. We then generalize the notion of deterministic regular expressions to regex, and show that the Glushkov construction can also be generalized. This allows us not only to efficiently decide the membership problem for deterministic regex, but also whether a regex is deterministic. After this, we study the expressive power of these models. Although deterministic regex share many of the limitations of deterministic regular expressions (in particular, the inherent non-determinism of some regular languages persists), their expressive power offers some surprises. Finally, we examine a subclass of deterministic regexes and DTMFA for which polynomial space minimization is possible, and we consider an alternative notion of determinism.

From the perspective of deterministic regular expressions, this paper proposes a natural extension that significantly increases the expressive power, while still having a tractable membership problem. From a regex point of view, we restrict regex to their deterministic core, thus obtaining a tractable subclass. Hence, the authors intend this paper as a starting point for further work, as it opens a new direction on research into making regex tractable. For space reasons, detailed proofs are given in a full version of the paper [21].

**Main contributions.** The main conceptual contribution of this paper are the notion of determinism in regex, and an appropriate deterministic automaton model. The main challenge from this point of view was finding a natural extension of deterministic regular expressions that preserves the following properties: A natural definition of determinism that can be checked efficiently and also has an automata-theoretic characterization, and an efficient Glushkov-style conversion to automata that decide the membership problem efficiently. Regarding technical contributions, the authors would like to emphasize that, in addition to the effort that was needed to accomplish the aforementioned goals, the paper uses subtleties of the back-reference operator in novel ways. By using these, deterministic regex can define non-deterministic regular languages (in particular, all unary regular languages), as well as infinite languages that are not pumpable in the usual sense.

**Related work.** Regex were first examined from a theoretical point of view by Aho [2], but without fully defining the semantics. There were various proposals for semantics, of which we mention the first by Câmpeanu, Salomaa, Yu [10], and the recent one by Schmid [35], which is the basis for this paper. Apart from defining the semantics, there was work on the expressive power [10, 11, 20], the static analysis [11, 18, 19], and the tractability of the membership problem (investigated in terms of a strongly restricted subclass of regex) [16, 17]. They have also been compared to related models in database theory, e. g. graph databases [4] and information extraction [15, 19].

Following the original paper by Brüggemann-Klein and Wood [9], deterministic regular expressions have been studied extensively. Aspects include computing the Glushkov au-

tomaton and deciding the membership problem (e.g. [8, 24, 34]), static analysis (cf. [31]), deciding whether a regular language is deterministic (e.g. [12, 24, 30]), closure properties and descriptive complexity [28], and learning (e.g. [5]). One noteworthy extension are counter operators (e.g. [23, 24, 27]), which we briefly address in Section 7.

## 2 Preliminaries

We use  $\varepsilon$  to denote the *empty word*. The subset and proper subset relation are denoted by  $\subseteq$  and  $\subset$ , respectively. Let  $\Sigma$  be a finite terminal alphabet. Unless otherwise noted, we assume  $|\Sigma| \geq 2$ . Let  $\Xi$  be an infinite variable alphabet with  $\Xi \cap \Sigma = \emptyset$ . Let  $w \in \Sigma^*$ , then, for every  $i$ ,  $1 \leq i \leq |w|$ ,  $w[i]$  denotes the symbol at position  $i$  of  $w$ . We define  $w^0 := \varepsilon$  and  $w^{i+1} := w^i \cdot w$  for all  $i \geq 0$ , and, for  $w = a_1 \cdots a_n$  with  $a_i \in \Sigma$ , let  $w^{m+\frac{i}{n}} = w^m \cdot a_1 \cdots a_i$  for all  $m \geq 0$  and all  $i$  with  $0 \leq i \leq n$ . A  $v \in \Sigma^*$  is a *factor* of  $w$  if there exist  $u_1, u_2 \in \Sigma^*$  with  $w = u_1 v u_2$ . If  $u_2 = \varepsilon$ ,  $v$  is also a *prefix* of  $w$ .

We use the notions of deterministic and non-deterministic finite automata (DFA and NFA) like [25]. If an NFA can have  $\varepsilon$ -transitions, we call it an  $\varepsilon$ -NFA. Given a class  $\mathcal{C}$  of language description mechanisms (e.g., a class of automata or regular expressions), we use  $\mathcal{L}(\mathcal{C})$  to denote the class of all languages  $\mathcal{L}(C)$  with  $C \in \mathcal{C}$ . The *membership problem* for  $\mathcal{C}$  is defined as follows: Given a  $C \in \mathcal{C}$  and a  $w \in \Sigma^*$ , is  $w \in \mathcal{L}(C)$ ?

### 2.1 Regex

► **Definition 1** (Syntax of regex). We define RX, the set of *regex* over  $\Sigma$  and  $\Xi$ , recursively: **Terminals and  $\varepsilon$ :**  $a \in \text{RX}$  and  $\text{var}(a) = \emptyset$  for every  $a \in (\Sigma \cup \{\varepsilon\})$ .

**Variable reference:**  $\&x \in \text{RX}$  and  $\text{var}(\&x) = \{x\}$  for every  $x \in \Xi$ .

**Concatenation:**  $(\alpha \cdot \beta) \in \text{RX}$  and  $\text{var}(\alpha \cdot \beta) = \text{var}(\alpha) \cup \text{var}(\beta)$  if  $\alpha, \beta \in \text{RX}$ .

**Disjunction:**  $(\alpha \vee \beta) \in \text{RX}$  and  $\text{var}(\alpha \vee \beta) = \text{var}(\alpha) \cup \text{var}(\beta)$  if  $\alpha, \beta \in \text{RX}$ .

**Kleene plus:**  $(\alpha^+) \in \text{RX}$  and  $\text{var}(\alpha^+) = \text{var}(\alpha)$  if  $\alpha \in \text{RX}$ .

**Variable binding:**  $\langle x : \alpha \rangle \in \text{RX}$  and  $\text{var}(\langle x : \alpha \rangle) = \text{var}(\alpha) \cup \{x\}$  if  $\alpha \in \text{RX}$  with  $x \in \Xi \setminus \text{var}(\alpha)$ .

In addition, we allow  $\emptyset$  as a regex (with  $\text{var}(\emptyset) = \emptyset$ ), but we do not allow  $\emptyset$  to occur in any other regex. An  $\alpha \in \text{RX}$  with  $\text{var}(\alpha) = \emptyset$  is called a *proper regular expression*, or just *regular expression*. We use REG to denote the set of all regular expressions.

We add and omit parentheses freely, as long as the meaning remains clear. We use the Kleene star  $\alpha^*$  as shorthand for  $\varepsilon \vee \alpha^+$ , and  $A$  as shorthand for  $\bigvee_{a \in A} a$  for non-empty  $A \subseteq \Sigma$ .

We define the semantics of regex using the *ref-words* (short for *reference words*) by Schmid [35]. A ref-word is a word over  $(\Sigma \cup \Xi \cup \Gamma)$ , where  $\Gamma := \{[_x, ]_x \mid x \in \Xi\}$ . Intuitively, the symbols  $[_x$  and  $]_x$  mark the beginning and the end of the match that is stored in the variable  $x$ , while an occurrence of  $x$  represents a reference to that variable. Instead of defining the language of a regex  $\alpha$  directly, we first treat  $\alpha$  as a generator of ref-words by defining its *ref-language*  $\mathcal{R}(\alpha)$ . If  $\alpha \in \Sigma \cup \{\varepsilon\}$ ,  $\mathcal{R}(\alpha) := \{\alpha\}$ ; and  $\mathcal{R}(\&x) := \{x\}$  for all  $x \in \Xi$ . Furthermore,  $\mathcal{R}(\alpha \cdot \beta) := \mathcal{R}(\alpha) \cdot \mathcal{R}(\beta)$ ,  $\mathcal{R}(\alpha \vee \beta) := \mathcal{R}(\alpha) \cup \mathcal{R}(\beta)$ , and  $\mathcal{R}(\alpha^+) := \mathcal{R}(\alpha)^+$ . Finally,  $\mathcal{R}(\langle x : \alpha \rangle) := ([_x \mathcal{R}(\alpha)]_x)$ . For regular expressions,  $\mathcal{L}(\alpha) = \mathcal{R}(\alpha)$ . Alternatively,  $\mathcal{R}(\alpha) := \mathcal{L}(\alpha_{\mathcal{R}})$ , where the proper regular expression  $\alpha_{\mathcal{R}}$  is obtained by replacing each sub-regex  $\langle x : \beta \rangle$  of  $\alpha$  with  $[_x \beta_{\mathcal{R}}]_x$ , and each  $\&x$  with  $x$ .

Intuitively speaking, every occurrence of a variable  $x$  in some  $r \in \mathcal{R}(\alpha)$  functions as a pointer to the next factor  $[_x v]_x$  to the left of this occurrence (or to  $\varepsilon$  if no such factor exists). In this way, a ref-word  $r$  compresses a word over  $\Sigma$ , the so-called dereference  $\mathcal{D}(r)$  of  $r$ , which can be obtained by replacing every variable occurrence  $x$  by the corresponding factor  $v$  (note

that  $v$  might again contain variable occurrences, which need to be replaced as well), and removing all symbols  $[x, ]_x \in \Gamma$  afterwards. See [35] for a more detailed definition, or the following Example 2 for an illustration. Finally, we define  $\mathcal{L}(\alpha) := \{\mathcal{D}(r) \mid r \in \mathcal{R}(\alpha)\}$ .

► **Example 2.** Let  $\alpha := ((x : (\mathbf{a} \vee \mathbf{b})^+) \& x)^+$ . Then  $\mathcal{R}(\alpha) = \{[xw_1]_x \cdot x \cdots [xw_n]_x \cdot x \mid n \geq 1, w_i \in \{\mathbf{a}, \mathbf{b}\}^+\}$ . Hence,  $\mathcal{L}(\alpha) = (L_{\text{copy}})^+$ , with  $L_{\text{copy}} := \{ww \mid w \in \{\mathbf{a}, \mathbf{b}\}^+\}$ . Let  $\alpha_{\text{sq}} := ((x : \& y) \langle y : \& x \cdot \mathbf{a} \rangle)^*$ . Then  $\mathcal{R}(\alpha_{\text{sq}}) = \{([xy]_x \cdot [yx \cdot \mathbf{a}]_y)^i \mid i \geq 0\}$ . For example, consider the ref word  $r_3 = [xy]_x \cdot [yx \cdot \mathbf{a}]_y \cdot [xy]_x \cdot [yx \cdot \mathbf{a}]_y \cdot [xy]_x \cdot [yx \cdot \mathbf{a}]_y$  with  $\mathcal{D}(r_3) = \mathbf{a}^9$ . Using induction, we can verify that  $\mathcal{D}(r_i) = \mathbf{a}^{i^2}$ . Thus,  $\mathcal{L}(\alpha_{\text{sq}}) = \{\mathbf{a}^{n^2} \mid n \geq 0\}$ .

Hence, unlike regular expressions, regex can define non-regular languages. The expressive power comes at a price: their membership problem is NP-complete (follows from Angluin [3]), and various other problems are undecidable (Freydenberger [18]). Starting with Aho [2], there have been various approaches to specifying syntax and semantics of regex. While [2] only sketched the intuition behind the semantics, the first formal definition (using parse trees) was proposed by Câmpeanu, Salomaa, Yu [10], followed by the ref-words of Schmid [35]. For a comparison between these approaches and actual implementations, see the full version [21].

### 3 Memory Automata with Trap State

*Memory automata* [35] are a simple automaton model that characterizes  $\mathcal{L}(\text{RX})$ . Intuitively speaking, these are classical finite automata that can record consumed factors in memories, which can be recalled later on in order to consume the same factor again. However, for our applications, we need to slightly adapt this model to *memory automata with trap-state*.

► **Definition 3.** For every  $k \in \mathbb{N}$ , a  $k$ -*memory automaton with trap-state*, denoted by  $\text{TMFA}(k)$ , is a tuple  $M = (Q, \Sigma, \delta, q_0, F)$ , where  $Q$  is a finite set of *states* that contains the *trap-state*  $[\text{trap}]$ ,  $\Sigma$  is a finite *alphabet*,  $q_0 \in Q$  is the *initial state*,  $F \subseteq Q$  is the set of *final states* and  $\delta: Q \times (\Sigma \cup \{\varepsilon\} \cup \{1, 2, \dots, k\}) \rightarrow \mathcal{P}(Q \times \{\mathbf{o}, \mathbf{c}, \mathbf{r}, \diamond\}^k)$  is the *transition function* (where  $\mathcal{P}(A)$  denotes the power set of a set  $A$ ), which satisfies  $\delta([\text{trap}], b) = \{([\text{trap}], \diamond, \diamond, \dots, \diamond)\}$ , for every  $b \in \Sigma \cup \{\varepsilon\}$ , and  $\delta([\text{trap}], i) = \emptyset$ , for every  $i, 1 \leq i \leq k$ . The elements  $\mathbf{o}$ ,  $\mathbf{c}$ ,  $\mathbf{r}$  and  $\diamond$  are called *memory instructions* (they stand for opening, closing and reseting a memory, respectively, and  $\diamond$  leaves the memory unchanged).

A *configuration* of  $M$  is a tuple  $(q, w, (u_1, r_1), \dots, (u_k, r_k))$ , where  $q \in Q$  is the *current state*,  $w$  is the *remaining input* and, for every  $i, 1 \leq i \leq k$ ,  $(u_i, r_i)$  is the *configuration of memory  $i$* , where  $u_i \in \Sigma^*$  is the *content of memory  $i$*  and  $r_i \in \{\mathbf{0}, \mathbf{C}\}$  is the *status of memory  $i$*  (i.e.,  $r_i = \mathbf{0}$  means that memory  $i$  is open and  $r_i = \mathbf{C}$  means that it is closed). The *initial configuration* of  $M$  (on input  $w$ ) is the configuration  $(q_0, w, (\varepsilon, \mathbf{C}), \dots, (\varepsilon, \mathbf{C}))$ , a configuration  $(q, w, (u_1, r_1), \dots, (u_k, r_k))$  is an *accepting configuration* if  $w = \varepsilon$  and  $q \in F$ .

$M$  can change from a configuration  $c = (q, vw, (u_1, r_1), \dots, (u_k, r_k))$  to a configuration  $c' = (p, w, (u'_1, r'_1), \dots, (u'_k, r'_k))$ , denoted by  $c \vdash_M c'$ , if there exists a transition  $\delta(q, b) \ni (p, s_1, \dots, s_k)$  with either  $(b \in (\Sigma \cup \{\varepsilon\})$  and  $v = b$ ) or  $(b \in \{1, 2, \dots, k\}, s_b = \mathbf{c}$  and  $v = u_b)$ , and, for every  $i, 1 \leq i \leq k$ ,

$$\begin{aligned} s_i = \diamond \wedge r_i = \mathbf{0} &\Rightarrow (u'_i, r'_i) = (u_i v, r_i), & s_i = \diamond \wedge r_i = \mathbf{C} &\Rightarrow (u'_i, r'_i) = (u_i, r_i), \\ s_i = \mathbf{o} &\Rightarrow (u'_i, r'_i) = (v, \mathbf{0}), & s_i = \mathbf{c} &\Rightarrow (u'_i, r'_i) = (u_i, \mathbf{C}), \\ s_i = \mathbf{r} &\Rightarrow (u'_i, r'_i) = (\varepsilon, \mathbf{C}). \end{aligned}$$

Furthermore,  $M$  can change from a configuration  $(q, vw, (u_1, r_1), \dots, (u_k, r_k))$  to the configuration  $([\text{trap}], w, (u_1, r_1), \dots, (u_k, r_k))$ , if  $\delta(q, b) \ni (p, s_1, \dots, s_k)$  for some  $p \in Q$ ,  $b \in \{1, 2, \dots, k\}$  and  $s_b = \mathbf{c}$ , such that  $u_b = vv'$  with  $v' \neq \varepsilon$  and  $v'[1] \neq w[1]$ .



A transition  $\delta(q, b) \ni (p, s_1, s_2, \dots, s_k)$  is an  $\varepsilon$ -transition if  $b = \varepsilon$  and is called *consuming*, otherwise (if all transitions are consuming, then  $M$  is called  $\varepsilon$ -free). If  $b \in \{1, 2, \dots, k\}$ , it is called a *memory recall transition* and the situation that a memory recall transition leads to the state [trap], is called a *memory recall failure*.

The symbol  $\vdash_M^*$  denotes the reflexive and transitive closure of  $\vdash_M$ . A  $w \in \Sigma^*$  is *accepted* by  $M$  if  $c_{\text{init}} \vdash_M^* c_f$ , where  $c_{\text{init}}$  is the initial configuration of  $M$  on  $w$  and  $c_f$  is an accepting configuration. The set of words accepted by  $M$  is denoted by  $\mathcal{L}(M)$ .

Note that executing the open action  $\circ$  on a memory that already contains some word discards the previous contents of that memory. For illustrations and examples for TMFA, we refer to [35]. A crucial part of TMFA is the trap-state [trap], in which computations terminate, if a memory recall failure happens. If [trap] is not accepting, then TMFA are (apart from negligible formal differences) identical to the memory automata introduced in [35], which characterize the class of regex language. If, on the other hand, [trap] is accepting, then every computation with a memory recall failure is accepting (independent from the remaining input). While it seems counter-intuitive to define the words of a language via “failed” back-references, the possibility of having an accepting trap-state yields closure under complement for *deterministic* TMFA (see Theorem 6). It will be convenient to consider the partition of TMFA into  $\text{TMFA}^{\text{rej}}$  and  $\text{TMFA}^{\text{acc}}$  (having a rejecting and an accepting trap-state, respectively).

Every  $\text{TMFA}^{\text{acc}}$  can be transformed into an equivalent  $\text{TMFA}^{\text{rej}}$ , which implies  $\mathcal{L}(\text{TMFA}) = \mathcal{L}(\text{TMFA}^{\text{rej}})$ ; thus, it follows from [35] that TMFA characterize  $\mathcal{L}(\text{RX})$ . The idea of this construction is as follows. Every memory  $i$  is simulated by two memories  $(i, 1)$  and  $(i, 2)$ , which store a (nondeterministically guessed) factorisation of the content of memory  $i$ . This allows us to guess and verify if a memory recall failure occurs, i. e.,  $(i, 1)$  stores the longest prefix that can be matched and  $(i, 2)$  starts with the first mismatch. For correctness, it is crucial that every possible factorisation of the content of a memory  $i$  can be guessed.

► **Theorem 4.**  $\mathcal{L}(\text{TMFA}) = \mathcal{L}(\text{TMFA}^{\text{rej}}) = \mathcal{L}(\text{RX})$ .

A consequence of the proof is that TMFA inherits the NP-hardness of the membership problem from RX. We do not devote more attention to this, as we focus on deterministic TMFA: A TMFA is *deterministic* (or a DTMFA, for short) if  $\delta$  satisfies  $|\delta(q, b)| \leq 1$ , for every  $q \in Q$  and  $b \in \Sigma \cup \{\varepsilon\} \cup \{1, 2, \dots, k\}$  (for the sake of convenience, we then interpret  $\delta$  as a partial function with range  $Q \times \{\circ, \mathbf{c}, \mathbf{r}, \diamond\}^k$ ), and, furthermore, for every  $q \in Q$ , if  $\delta(q, x)$  is defined for some  $x \in \{1, 2, \dots, k\} \cup \{\varepsilon\}$ , then, for every  $y \in (\Sigma \cup \{\varepsilon\} \cup \{1, 2, \dots, k\}) \setminus \{x\}$ ,  $\delta(q, y)$  is undefined. Analogously to TMFA, we partition DTMFA into  $\text{DTMFA}^{\text{acc}}$  and  $\text{DTMFA}^{\text{rej}}$ .

The algorithmically most important feature of DTMFA is that their membership can be solved efficiently by running the automaton on the input word. However, for each processed input symbol, there might be a delay of at most  $|Q|$  steps, due to  $\varepsilon$ -transitions and recalls of empty memories, which leads to  $O(|Q||w|)$ . Removing such non-consuming transitions first, is possible, but problematic. In particular, recalls of empty memories depend on the specific input word and could only be determined beforehand by storing for each memory whether it is empty, which is too expensive. However, by  $O(|Q|^2)$  preprocessing, we can compute the information that is needed in order to determine in  $O(k)$  where to jump if certain memories are empty, and which memories are currently empty can be determined on-the-fly while processing the input. This leads to a delay of only  $k$ , the number of memories:

► **Theorem 5.** *Given  $M \in \text{DTMFA}$  with  $n$  states and  $k$  memories, and  $w \in \Sigma^*$ , we can decide in time  $O(n^2 + k|w|)$ , whether or not  $w \in \mathcal{L}(M)$ .*

Note that the preprocessing in the proof of Theorem 5 is only required once, so we can solve the membership for several words  $w_i$  in  $O(n^2 + k \sum |w_i|)$ . Moreover, if it is guaranteed that no empty memories are recalled, then membership can be solved in  $O(n + |w|)$  (where  $O(n)$  is needed in order to remove  $\varepsilon$ -transitions).

Similar to DFA, it is possible to complement DTMFA by toggling the acceptance of states. However, for DTMFA, we have to remove  $\varepsilon$ -transitions and recalls of empty memories. In particular, the construction for Theorem 6 uses the finite control to store whether memories are empty or not, which causes a blow-up that is exponential in the number of memories.

► **Theorem 6.**  $\mathcal{L}(\text{DTMFA})$  is closed under complement.

We next discuss expressive power: If there is a constant upper bound on the lengths of contents of memories that are recalled in accepting computations of an  $M \in \text{DTMFA}$ , then memories can be simulated by the finite state control; thus,  $\mathcal{L}(M) \in \mathcal{L}(\text{REG})$ . Consequently, if  $\mathcal{L}(M) \notin \mathcal{L}(\text{REG})$ , there is a word  $uvw$  that is accepted by recalling some memory with an arbitrarily large content  $v$ . Moreover, if [trap] is non-accepting, then no word can be accepted that contains  $u$  as a prefix, but not  $w$ , since this will cause a memory recall failure. Intuitively speaking, a  $\text{DTMFA}^{\text{rej}}$  for a non-regular language makes arbitrarily large “jumps”:

► **Lemma 7 (Jumping Lemma).** Let  $L \in \mathcal{L}(\text{DTMFA}^{\text{rej}})$ . Then either  $L$  is regular, or for every  $m \geq 0$ , there exist  $n \geq m$  and  $p_n, v_n \in \Sigma^+$  such that

1.  $|v_n| = n$ ,
2.  $v_n$  is a factor of  $p_n$ ,
3.  $p_n v_n$  is a prefix of a word from  $L$ ,
4. for all  $u \in \Sigma^+$ ,  $p_n u \in L$  only if  $v_n$  is a prefix of  $u$ .

► **Example 8.** Let  $L := \{ww \mid w \in \Sigma^*\}$  with  $|\Sigma| \geq 2$ , which is well-known to be not regular. Assume  $L \in \mathcal{L}(\text{DTMFA}^{\text{rej}})$  and choose  $m := 1$ . Then there exist  $n \geq 1$  and  $p_n, v_n \in \Sigma^*$  that satisfy the conditions of Lemma 7. Choose  $a \in \Sigma$  that is not the first letter of  $v_n$ , and define  $u := ap_n a$ . Then  $v_n$  is not a prefix of  $u$ , but  $p_n u = (p_n a)^2 \in L$ , which is a contradiction.

► **Example 9.** Let  $L := \{a^i b a^j \mid i > j \geq 0\}$ . Using textbook methods, it is easily shown that  $L$  is not regular. Now, assuming that  $L \in \mathcal{L}(\text{DTMFA}^{\text{rej}})$ , choose  $m := 4$ . Then there exist  $n \geq 4$  and  $p_n, v_n \in \Sigma^+$  that satisfy the conditions of Lemma 7. As  $p_n v_n$  is a prefix of a word in  $L$ , either  $p_n = a^i$  or  $p_n = a^i b a^j$  with  $i, j \geq 0$  (and  $i \geq 4$  or  $i + j \geq 3$ ). In the first case, consider  $u := ba$ . Then  $p_n u = a^i b a$  with  $i \geq 4$ ; hence,  $p_n u \in L$ . But  $u$  starts with  $b$ , and  $v_n$  is a factor of  $p_n = a^i$ . Contradiction, as  $v_n$  cannot be a prefix of  $u$ . For the second case, let  $u := a$ . As  $p_n v_n$  is a prefix of a word in  $L$ , and as  $|v_n| = n$ ,  $i > j + n \geq j + 4$  must hold. Hence,  $p_n u = a^i b a^{j+1}$ , and  $p_n u \in L$ . Contradiction, as  $v_n$  is not a prefix of  $u$ .

For unary languages, there is an alternative to Lemma 7 that is easier to apply and characterizes unary  $\text{DTMFA}^{\text{rej}}$ -languages. It is built on the following definition: A language  $L \subseteq \{a\}^*$  is an *infinite arithmetic progression* if  $L = \{a^{bi+c} \mid i \geq 0\}$  for some  $b \geq 1$ ,  $c \geq 0$ .

► **Lemma 10.** Let  $L \in \mathcal{L}(\text{DTMFA}^{\text{rej}})$  be an infinite language with  $L \subseteq \{a\}^*$ . The following conditions are equivalent:

1.  $L$  is regular.
2.  $L$  contains an infinite arithmetic progression.
3. There is  $b \geq 1$  such that, for every  $n \geq 0$ ,  $a^{bi+c_n} \in L$  for some  $c_n \geq 0$  and all  $0 \leq i \leq n$ .

► **Example 11.** Let  $\alpha := (x : aa^+)(\&x)^+$  (this regex is also known as “Abigail’s expression” [1] in the PERL community). Then  $\mathcal{L}(\alpha) = \{a^{mn} \mid m, n \geq 2\}$ . In other words,  $\alpha$  generates the language of all  $a^i$  such that  $i$  is a composite number (i. e., not a prime number). As  $\mathcal{L}(\alpha)$  is not regular and contains the arithmetic progression  $2i + 4$ , Lemma 10 yields  $\mathcal{L}(\alpha) \notin \mathcal{L}(\text{DTMFA}^{\text{rej}})$ .

The following result is a curious consequence of Lemma 10:

► **Proposition 12.** *Over unary alphabets,  $\mathcal{L}(\text{DTMFA}^{\text{rej}}) \cap \mathcal{L}(\text{DTMFA}^{\text{acc}}) = \mathcal{L}(\text{REG})$ .*

## 4 Deterministic Regex

In order to define deterministic regex as an extension of deterministic regular expressions, we first extend the notion of a *marked alphabet* that is commonly used for the latter: For every alphabet  $A$ , let  $\tilde{A} := \{a_{(n)} \mid a \in A, n \geq 1\}$ . For every  $\alpha \in \text{RX}$ , we define  $\tilde{\alpha}$  as a regex that is obtained by taking  $\alpha_{\mathcal{R}}$  (the proper regular expression over  $\Sigma \cup \Xi \cup \Gamma$  that generates the ref-language  $\mathcal{R}(\alpha)$ ), and marking each occurrence of  $\chi \in (\Sigma \cup \Xi \cup \Gamma)$  by a unique number (to make this well-defined, we assume that the markings start at 1 and are increased stepwise). For example, if  $\alpha := \langle y : (\mathbf{a} \vee \&x)^* \cdot (\varepsilon \vee \mathbf{b} \cdot \mathbf{a}) \rangle \cdot \&y$ , then  $\tilde{\alpha} = [_{y(1)}(\mathbf{a}_{(2)} \vee x_{(3)})^* \cdot (\varepsilon \vee \mathbf{b}_{(4)} \cdot \mathbf{a}_{(5)})]_{y(6)} \cdot y_{(7)}$ . We also use these markings in the ref-words: For example,  $[_{y(1)}\mathbf{a}_{(2)}\mathbf{a}_{(2)}x_{(3)}\mathbf{a}_{(2)}]_{y(6)}y_{(7)} \in \mathcal{R}(\tilde{\alpha})$ .

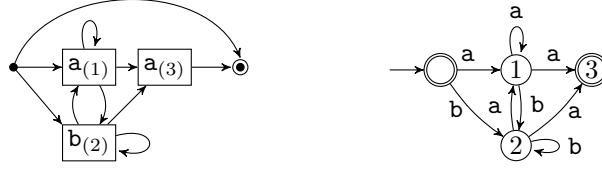
Before we explain this definition and use it to define deterministic regex, we first discuss the special case of deterministic regular expressions: A proper regular expression  $\alpha$  is *not* deterministic if there exist words  $u, v_1, v_2 \in \tilde{\Sigma}^*$ , a terminal  $a \in \Sigma$  and positions  $i \neq j$  such that  $ua_{(i)}v_1$  and  $ua_{(j)}v_2$  are elements of  $\mathcal{L}(\tilde{\alpha})$  (see e. g. [9, 24]). Otherwise, it is a *deterministic proper regular expression* (or, for short, just *deterministic regular expression*).

The intuition behind this definition is based on the Glushkov construction for the conversion of regular expressions into finite automata, as a regular expression  $\alpha$  is deterministic if and only if its *Glushkov automaton*  $\mathcal{M}(\alpha)$  is deterministic. Given a regular expression  $\alpha$ , we define  $\mathcal{M}(\alpha)$  in the following way: First, we use the marked regular expression  $\tilde{\alpha}$  to construct its *occurrence graph*<sup>1</sup>  $G_{\tilde{\alpha}}$ , a directed graph that has a source node `src`, a sink node `snk`, and one node for each  $a_{(i)}$  in  $\tilde{\alpha}$ . The edges are constructed in the following way: Each node  $a_{(i)}$  has an incoming edge from `src` if  $a_{(i)}$  can be the first letter of a word in  $\mathcal{L}(\tilde{\alpha})$ , and an outgoing edge to `snk` if it can be the last letter of such a word. Furthermore, for each factor  $a_{(i)}b_{(j)}$  that occurs in a word of  $\mathcal{L}(\tilde{\alpha})$ , there is an edge from  $a_{(i)}$  to  $b_{(j)}$ . As a consequence, there is a one-to-one-correspondence between marked words in  $\mathcal{L}(\tilde{\alpha})$  and paths from `src` to `snk` in  $G_{\tilde{\alpha}}$ . To obtain  $\mathcal{M}(\alpha)$ , we directly interpret  $G_{\tilde{\alpha}}$  as NFA over  $\Sigma$ : The source `src` is the starting state, each node  $a_{(i)}$  is a state  $q_i$ , and an edge from  $a_{(i)}$  to  $b_{(j)}$  corresponds to a transition from  $q_i$  to  $q_j$  when reading  $b$ . The sink `snk` does not become a state; instead, each node with an edge to `snk` is a final state (hence,  $\mathcal{M}(\alpha)$  contains the source state, and one state for every terminal in  $\alpha$ ). This interpretation allows us to treat occurrence graphs as an alternative notation for a subclass of NFA (namely those where the starting state is not reachable from other states, and for each state  $q$ , there is a characteristic terminal  $a_q$  such that all transitions to  $q$  read  $a_q$ ). When doing so, we usually omit the occurrence markings on the nodes in graphical representations.

Intuitively,  $\mathcal{M}(\alpha)$  treats each terminal of  $\alpha$  as a state. Recall that  $\alpha$  is not deterministic if there exists words  $ua_{(i)}v_1$  and  $ua_{(j)}v_2$  in  $\mathcal{L}(\tilde{\alpha})$  with  $i \neq j$ . This corresponds to the situation where, after reading  $u$ ,  $\mathcal{M}(\alpha)$  has to decide between states  $a_{(i)}$  and  $a_{(j)}$  for the input letter  $a$ .

► **Example 13.** Let  $\alpha := \langle \varepsilon \vee ((\mathbf{a} \vee \mathbf{b})^+ \mathbf{a}) \rangle$ . Then  $\tilde{\alpha} = \langle \varepsilon \vee ((\mathbf{a}_1 \vee \mathbf{b}_2)^+ \mathbf{a}_3) \rangle$ , and  $\mathcal{M}(\alpha)$ , the Glushkov automaton of  $\alpha$ , is defined as follows:

<sup>1</sup> Most literature, like [9], defines the occurrence graph only implicitly by using sets `first`, `last`, and `follow`, which correspond to the edge from `src`, the edges to `snk`, or to the other edges of the graph, respectively. The explicit use of a graph is taken from the  $k$ -occurrence automata by Bex et al. [5]. We shall see that an advantage of graphs is that they can be easily extended by describing memory actions to the edges.



To the left,  $\mathcal{M}(\alpha)$  is represented as an occurrence graph, to the right in standard NFA notation. Then  $\mathcal{M}(\alpha)$  and  $\alpha$  are both not deterministic: For  $\mathcal{M}(\alpha)$ , consider state 1; for  $\alpha$ , consider  $u = \mathbf{a}_{(1)}$ ,  $v_1 = \mathbf{a}_{(3)}$ ,  $v_2 = \varepsilon$ , and the words  $u\mathbf{a}_{(1)}v_1$  and  $u\mathbf{a}_{(3)}v_2$ .

As shown in [9],  $\mathcal{L}(\text{DREG}) \subset \mathcal{L}(\text{REG})$  (also see [12, 30], or Lemma 23 below). Like for determinism of regular expressions, the key idea behind our definition of deterministic regex is that a matcher for the expression treats terminals (and variable references) as states. Then an expression is deterministic if the current symbol of the input word always uniquely determines the next state and all necessary variable actions. For regular expressions, non-determinism can only occur when the matcher has to decide between two occurrences of the same terminal symbol; but as regex also need to account for non-determinism that is caused by variable operations or references, their definition of non-determinism is more complicated.

► **Definition 14.** An  $\alpha \in \text{RX}$  is *not deterministic* if there exist  $\rho_1, \rho_2 \in \mathcal{R}(\tilde{\alpha})$  such that any of the following conditions is met for some  $r, s_1, s_2 \in (\tilde{\Sigma} \cup \tilde{\Xi} \cup \tilde{\Gamma})^*$  and  $\gamma_1, \gamma_2 \in \tilde{\Gamma}^*$ :

1.  $\rho_1 = r \cdot \gamma_1 \cdot a_{(i)} \cdot s_1$  and  $\rho_2 = r \cdot \gamma_2 \cdot a_{(j)} \cdot s_2$  with  $a \in \Sigma$  and  $i \neq j$ ,
2.  $\rho_1 = r \cdot \gamma_1 \cdot x_{(i)} \cdot s_1$  and  $\rho_2 = r \cdot \gamma_2 \cdot \chi_{(j)} \cdot s_2$  with  $x \in \Xi$ ,  $\chi \in (\Sigma \cup \Xi)$  and  $i \neq j$ ,
3.  $\rho_1 = r \cdot \gamma_1 \cdot \chi_{(i)} \cdot s_1$  and  $\rho_2 = r \cdot \gamma_2 \cdot \chi_{(i)} \cdot s_2$  with  $\chi \in (\Sigma \cup \Xi)$  and  $\gamma_1 \neq \gamma_2$ ,
4.  $\rho_1 = r \cdot \gamma_1$  and  $\rho_2 = r \cdot \gamma_2$  with  $\gamma_1 \neq \gamma_2$ .

Otherwise,  $\alpha$  is *deterministic*. We use  $\text{DRX}$  to denote the set of all deterministic regex, and define  $\text{DREG} := \text{DRX} \cap \text{REG}$  as the set of deterministic regular expressions.

► **Example 15.** Let  $\alpha_1 := (\langle x: \mathbf{a} \rangle \vee \mathbf{a})$ ,  $\alpha_2 := (\mathbf{a} \vee \&x)$ ,  $\alpha_3 := (\langle x: \varepsilon \rangle \vee \varepsilon)\mathbf{a}$ ,  $\alpha_4 := (\langle x: \varepsilon \rangle \vee \varepsilon)$ . None of these regex are deterministic, as each  $\alpha_i$  meets the  $i$ -th condition of Definition 14. We discuss this for  $\alpha_1$ : Observe  $\tilde{\alpha}_1 = ([x_{(1)}\mathbf{a}_{(2)}]_{x_{(3)}}) \vee \mathbf{a}_{(4)}$ . Then choosing  $\rho_1 = [x_{(1)}\mathbf{a}_{(2)}]_{x_{(3)}}$  and  $\rho_2 = \mathbf{a}_{(4)}$ , with  $r = \varepsilon$ ,  $\gamma_1 = [x_{(1)}, s_1 = ]_{x_{(3)}}$ , and  $\gamma_2 = s_2 = \varepsilon$  shows the condition is met.

Let  $\beta_1 := \langle x: (\mathbf{a} \vee \mathbf{b})^* \rangle \mathbf{c} \cdot \&x$  and  $\beta_2 := (\langle x: \&xy \rangle \langle y: \&x \cdot \mathbf{a} \rangle)^*$ . Both regex are deterministic, with  $\mathcal{L}(\beta_1) := \{w\mathbf{c}w \mid w \in \{\mathbf{a}, \mathbf{b}\}^*\}$  and  $\mathcal{L}(\beta_2) = \{\mathbf{a}^{n^2} \mid n \geq 0\}$  (see Example 2).

Condition 1 of Definition 14 describes cases where non-determinism is caused by two occurrences of the same terminal ( $\gamma_1$  and  $\gamma_2$  are included for cases like  $\alpha_1$  in Example 15). If restricted to regular expressions, it is equivalent to the usual definition of deterministic regular expressions. Condition 2 expresses that the matcher has to decide between a variable reference and any other symbol; while in condition 3, the symbol is unique, but there is a non-deterministic choice between variable operations. Finally, condition 4 describes cases where the behaviour of variables is non-deterministic after the end of the word (while one could consider this edge case deterministic, this choice simplifies recursive definitions). In conditions 3 and 4, the definition not only requires that it is clear which variables are reset, but also that it is clear which part of the regex acts on the variables. Hence,  $(\langle x: \varepsilon \rangle \vee \langle x: \varepsilon \rangle)$  is also not deterministic. This is similar to the notion of strong determinism for regular expressions, see [23]. As one might expect, some non-deterministic regexes define  $\text{DRX}$ -languages:

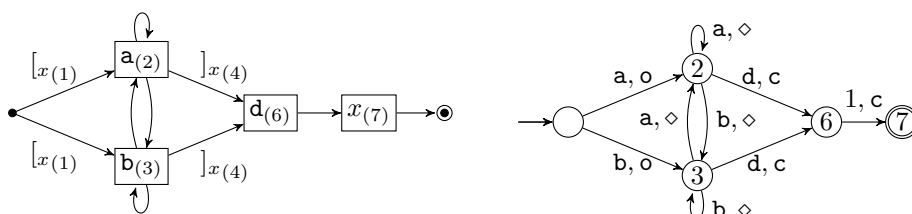
► **Example 16.** Let  $\Sigma = \{0, 1\}$  and  $\alpha := 1^+ \langle x: 0^* \rangle (1^+ \&x)^* 1^+$ . This regex was introduced by Fagin et al. [15], who call its language the “uniform-0-chunk language”. Obviously,  $\alpha$  is not deterministic (in fact, it satisfies conditions 1, 2, and 3 of Definition 14). Nonetheless, it is possible to express  $\mathcal{L}(\alpha)$  with the deterministic regex  $1(1^+ \vee (0 \langle x: 0^* \rangle 1^+ (0 \cdot \&x \cdot 1^+)^*))$ .

We now discuss the conversion from DRX to DTMFA<sup>rej</sup>, which generalizes the Glushkov construction of  $\mathcal{M}(\alpha)$  for regular expressions. The core idea is extending the occurrence graph to a *memory occurrence graph*  $G_{\tilde{\alpha}}$ , which has two crucial differences: First, instead of only considering terminals, each terminal and each variable reference of a regex  $\alpha$  becomes a node. Second, each edge is labelled with a ref-word from  $\tilde{\Gamma}^*$  that describes the memory actions (hence, there can be multiple edges from one node to another). In analogy to the occurrence graph, each memory occurrence graph can be directly interpreted as an  $\varepsilon$ -free TMFA<sup>rej</sup>.

► **Theorem 17.** *Let  $\alpha \in \text{RX}$ , and let  $n$  denote the number of occurrences of terminals and variable references in  $\alpha$ . We can construct an  $n+2$  state TMFA<sup>rej</sup>  $\mathcal{M}(\alpha)$  with  $\mathcal{L}(\mathcal{M}(\alpha)) = \mathcal{L}(\alpha)$  that is deterministic if and only if  $\alpha$  is deterministic. In time  $O(|\Sigma||\alpha|n)$ , the algorithm either*

1. *computes  $\mathcal{M}(\alpha)$  if  $\alpha$  is deterministic, or*
2. *detects that  $\alpha$  is not deterministic.*

► **Example 18.** Consider the deterministic regex  $\alpha := \langle x : (\mathbf{a} \vee \mathbf{b})^+ \rangle \cdot \mathbf{d} \cdot \&x$ . Applying the markings yields  $\tilde{\alpha} := [x_{(1)}(\mathbf{a}_{(2)} \vee \mathbf{b}_{(3)})^+]_{x_{(4)}} \cdot \mathbf{d}_{(6)} \cdot x_{(7)}$ , and  $\mathcal{M}(\alpha)$  is the following automaton:



To the left,  $\mathcal{M}(\alpha)$  is represented as the memory occurrence graph  $G_{\tilde{\alpha}}$ , to the right as the DTMFA that can be directly derived from this graph (which uses memory 1 for  $x$ ).

The construction from the proof of Theorem 17 behaves like the Glushkov construction for regular expressions, with one important difference: On regex that are not deterministic, its running time may be exponential in the number of variables; as there are non-deterministic regex where conversion into a TMFA without  $\varepsilon$ -transitions requires an exponential amount of transitions. E. g., for  $k \geq 1$ , let  $\alpha := \mathbf{a} \cdot (\varepsilon \vee \langle x_1 : \varepsilon \rangle) \cdot \dots \cdot (\varepsilon \vee \langle x_k : \varepsilon \rangle) \cdot \mathbf{b}$  and  $\beta := \mathbf{a}(\bigvee_{1 \leq i \leq k} \langle x_i : \varepsilon \rangle)^* \mathbf{b}$ . An automaton that is derived with a Glushkov style conversion then contains states  $q_1$  and  $q_2$  that correspond to the terminals; and between these two states, there must be  $2^k$  different transitions to account for all possible combinations of actions on the variables. This suggests that converting a regex into a TMFA without  $\varepsilon$ -edges is only efficient for deterministic regex; while in general, it is probably advisable to use a construction with  $\varepsilon$ -edges.

By combining Theorems 17 and 5, due to  $n \leq |\alpha|$ , we immediately obtain the following:

► **Theorem 19.** *Given  $\alpha \in \text{DRX}$  with  $n$  occurrences of terminal symbols or variable references and  $k$  variables, and  $w \in \Sigma^*$ , we can decide in time  $O(|\Sigma||\alpha|n + k|w|)$ , whether  $w \in \mathcal{L}(\alpha)$ .*

If we ensure that recalled variables never contain  $\varepsilon$  (or that only a bounded number of variables references are possible in a row), we can even drop the factor  $k$ . For comparison, the membership problem for DREG can be decided in time  $O(|\Sigma||\alpha| + |w|)$  when using optimized versions of the Glushkov construction (see [8, 34]), and in  $O(|\alpha| + |w| \cdot \log \log |\alpha|)$  with the algorithm by Groz, Maneth, and Staworko [24] that does not compute an automaton.

## 5 Expressive Power

While Câmpeanu, Salomaa, Yu [10] and Carle and Narendran [11] state pumping lemmas for a class of regex, these do not apply to regex as defined in this paper. However, Lemmas 7 and 10, introduced in Section 3, shall be helpful for proving inexpressibility. A consequence of Lemma 10 is that there are infinite unary  $\text{DTMFA}^{\text{rej}}$ -languages that are not pumpable (in the sense that certain factors can be repeated arbitrarily often), as this would always lead to an arithmetic progression. It is also possible to demonstrate this phenomenon on larger alphabets, without relying on a trivial modification of the unary case:

► **Example 20.** The Fibonacci word  $F_\omega$  is the infinite word that is the limit of the sequence of words  $F_0 := \mathbf{b}$ ,  $F_1 := \mathbf{a}$ , and  $F_{n+2} := F_{n+1} \cdot F_n$  for all  $n \geq 0$ . The Fibonacci word has a number of curious properties. In particular, it includes no cubes (i. e., factors  $www$ , with  $w \neq \varepsilon$ ). This and various other properties are explained throughout Lothaire [29]. Let

$$\alpha := \mathbf{a}\langle x_0 : \mathbf{b} \rangle \langle x_1 : \mathbf{a} \rangle (\langle x_2 : \&x_1 \&x_0 \rangle \langle x_3 : \&x_1 \&x_0 \&x_1 \rangle \langle x_0 : \&x_3 \&x_2 \rangle \langle x_1 : \&x_3 \&x_2 \&x_3 \rangle)^*.$$

Then  $\mathcal{L}(\alpha) = \{F_{4i+3} \mid i \geq 0\}$ . Hence, the words of  $\mathcal{L}(\alpha)$  converge towards  $F_\omega$ . The proof of this equivalence is straightforward, but long. It uses that  $F_{n+3} = F_{n+1} \cdot F_n \cdot F_{n+1}$  holds for all  $n \geq 0$ . As  $F_\omega$  contains no cube, the same applies to all  $F_n$ . Thus,  $\mathcal{L}(\alpha)$  is a DRX-language that cannot be pumped by repeating factors of sufficiently large words arbitrarily often.

For further separations, we use the following language:

► **Example 21.** Let  $\alpha := \mathbf{a}^2 \cdot \langle x : \mathbf{a}^2 \rangle \cdot (\langle y : \&x \cdot \&x \rangle \cdot \langle x : \&y \cdot \&y \rangle)^*$ . Then  $\mathcal{L}(\alpha) = \{\mathbf{a}^{4^i} \mid i \geq 1\}$ .

From this, we define an  $L \in \mathcal{L}(\text{TMFA})$  with neither  $L \in \mathcal{L}(\text{DTMFA}^{\text{rej}})$ , nor  $L \in \mathcal{L}(\text{DTMFA}^{\text{acc}})$ :

► **Lemma 22.** Let  $L := \{\mathbf{a}^{4i+1} \mid i \geq 0\} \cup \{\mathbf{a}^{4^i} \mid i \geq 1\}$ . Then  $L \in \mathcal{L}(\text{TMFA}) \setminus \mathcal{L}(\text{DTMFA})$ .

While  $\text{DTMFA}^{\text{rej}}$ -inexpressibility provides us with a powerful sufficient criterion for DRX-inexpressibility, it is not powerful enough to cover all cases of DRX-inexpressibility. In particular, there are even regular languages that are no DRX-languages:

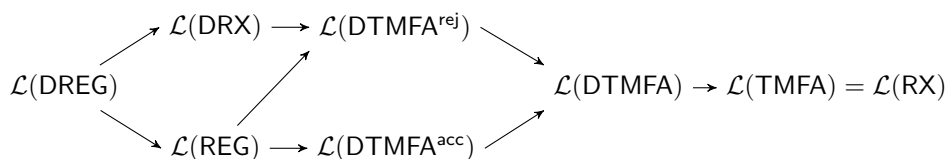
► **Lemma 23.** Let  $L := \mathcal{L}((\mathbf{ab})^*(\mathbf{a} \vee \varepsilon)) = \{(\mathbf{ab})^{\frac{1}{2}i} \mid i \geq 0\}$ . Then  $L \in \mathcal{L}(\text{REG}) \setminus \mathcal{L}(\text{DRX})$ .

The language  $L$  from Lemma 23 is also known to be a non-deterministic regular language (see e. g. [9]). Our proof can be seen as taking the idea behind the characterization of deterministic regular languages from [9], applying it to the specific language  $L$ , and also taking variables into account. While this accomplishes the task of proving that deterministic regex share some of the limitations of deterministic regular expressions, the approach does not generalize (at least not in a straightforward manner). In particular, deterministic regex can express regular languages that are not deterministic regular, and are also quite similar to  $L$ :

► **Example 24.** Let  $L := \{(\mathbf{ab})^{\frac{3}{2}i} \mid i \geq 0\}$ . Then  $L$  is generated by the non-deterministic regular expression  $(\mathbf{ababab})^*(\varepsilon \vee (\mathbf{aba}))$ , and one can show that  $L$  is not a deterministic regular language by using the BKW-algorithm [9] (also [12, 30]) on the minimal DFA  $M$  for  $L$ . But for  $\alpha := \mathbf{a}\langle y : \mathbf{b} \rangle \langle x : \mathbf{a} \rangle (\langle z : \&y \rangle \langle y : \&x \rangle \langle x : \&z \rangle)^*$ ,  $\alpha \in \text{DRX}$  and  $\mathcal{L}(\alpha) = L$ .

The “shifting gadget” that is used in Example 24 can be extended to show a far more general result for unary languages. Considering that  $\mathcal{L}(\text{DREG}) \subset \mathcal{L}(\text{REG})$  holds even over unary alphabets (cf. Losemann et al. [28]), the following result might seem surprising:

► **Theorem 25.** For every regular language  $L$  over a unary alphabet,  $L \in \mathcal{L}(\text{DRX})$ .



■ **Figure 1** The proper inclusions from Theorem 26. Arrows point from sub- to superset.

As a DFA with  $n$  states is converted into a deterministic regex of length  $O(n)$ , this construction is even efficient. We summarize our observations (also see Figure 1):

► **Theorem 26.**  $\mathcal{L}(\text{DREG}) \subset \mathcal{L}(\text{DRX}) \subset \mathcal{L}(\text{DTMFA}^{\text{rej}}) \subset \mathcal{L}(\text{DTMFA}) \subset \mathcal{L}(\text{TMFA}) = \mathcal{L}(\text{RX})$ .

The following pairs of classes are incomparable:  $\mathcal{L}(\text{DRX})$  and  $\mathcal{L}(\text{REG})$ ,  $\mathcal{L}(\text{DRX})$  and  $\mathcal{L}(\text{DTMFA}^{\text{acc}})$ , as well as  $\mathcal{L}(\text{DTMFA}^{\text{rej}})$  and  $\mathcal{L}(\text{DTMFA}^{\text{acc}})$ .

We can also use the examples from this section to show that  $\mathcal{L}(\text{DRX})$  and  $\mathcal{L}(\text{DTMFA}^{\text{rej}})$  are not closed under most of the commonly studied operations on languages:

► **Theorem 27.**  $\mathcal{L}(\text{DRX})$  and  $\mathcal{L}(\text{DTMFA}^{\text{rej}})$  are not closed under the following operations: union, concatenation, reversal, complement, homomorphism, and inverse homomorphism.  $\mathcal{L}(\text{DRX})$  is also not closed under intersection, and intersection with DREG-languages.

We leave open whether  $\mathcal{L}(\text{DTMFA}^{\text{rej}})$  is closed under intersection (with itself or with  $\mathcal{L}(\text{DREG})$ ), but we conjecture that this is not the case. We also leave open whether  $\mathcal{L}(\text{DRX})$  and  $\mathcal{L}(\text{DTMFA}^{\text{rej}})$  are closed under Kleene plus or star.

## 6 Two Variants of Determinism

In this section, we examine a restriction and an extension of DRX and DTMFA. We begin with the restriction, which we motivate with the following observation: As shown by Carle and Narendran [11], the intersection problem for regex is undecidable. For DRX, that proof cannot be used, but the result still holds (and by Theorem 17, this extends to DTMFA):

► **Theorem 28.** Given  $\alpha, \beta \in \text{DRX}$ , it is undecidable whether  $\mathcal{L}(\alpha) \cap \mathcal{L}(\beta) = \emptyset$ .

As a consequence, DTMFA intersection emptiness problem is also undecidable. Theorem 28 applies even to very restricted DRX, as no variable binding contains a reference to another variable,  $|\text{var}(\alpha)| = 2$ , and  $|\text{var}(\beta)| = 3$ . Hence, bounding the number of variables does not make the problem decidable. Instead, the key part seems to be that the variables occur under Kleene stars, which means that they can be reassigned an unbounded amount of times. Following similar observations, Freydenberger and Holldack [20] introduced the following concept: A regex is *variable-star-free* (*vstar-free*) if each of its plussed sub-regexes contains neither variable references, nor variable bindings. Analogously, we call a TMFA *memory-cycle-free* if it contains no cycle with a *memory transition* (a transition in a TMFA that is a memory recall, or that contains memory actions other than  $\diamond$ ). Let  $\text{RX}_{\text{vsf}}$  be the set of all vstar-free regex, and  $\text{DRX}_{\text{vsf}} = \text{RX}_{\text{vsf}} \cap \text{DRX}$ . Let  $\text{TMFA}_{\text{mcf}}$  be the set of all memory-cycle-free TMFA, and define  $\text{DTMFA}_{\text{mcf}}$ ,  $\text{TMFA}_{\text{mcf}}^{\text{rej}}$ , ... analogously. The proof of Theorem 17 allows us to conclude that  $\mathcal{M}(\alpha) \in \text{DTMFA}_{\text{mcf}}$  holds for every  $\alpha \in \text{DRX}_{\text{vsf}}$ . Likewise, we can use the proof of Theorem 4 to conclude  $\mathcal{L}(\text{TMFA}_{\text{mcf}}) = \mathcal{L}(\text{RX}_{\text{vsf}})$ . Note that for  $\varepsilon$ -free  $\text{DTMFA}_{\text{mcf}}$ , the membership problem can be decided in time  $O(|Q| + |w|)$ , as the preprocessing step of Theorem 5 is not necessary (as only a bounded number of variable references is possible in each run). Likewise, we can drop the factor  $k$  from Theorem 19 when restricted to  $\text{DRX}_{\text{vsf}}$ .

As shown by Freydenberger [19], it is decidable in PSPACE whether  $\bigcap_{i=1}^n \mathcal{L}(\alpha_i) = \emptyset$  for  $\alpha_1, \dots, \alpha_n \in \text{RX}_{\text{vsf}}$ . By combining the proof for this with some ideas from another construction from [19], we encode the intersection emptiness problem for  $\text{TMFA}_{\text{mcf}}$  in the *existential theory of concatenation with regular constraints* (a PSPACE-decidable, positive logic on words, see Diekert [13], Diekert, Jež, Plandowski [14]). This yields the following:

► **Theorem 29.** *Given  $M_1, \dots, M_n \in \text{TMFA}_{\text{mcf}}$ , we can decide whether  $\bigcap_{i=1}^n \mathcal{L}(M_i) = \emptyset$  in PSPACE. The problem is PSPACE-hard, even if restricted to  $\mathcal{L}(\alpha) \cap \mathcal{L}(\beta)$ ,  $\alpha \in \text{DRX}_{\text{vsf}}$  and  $\beta \in \text{DREG}$  (if the size of  $\Sigma$  is not bounded), or to  $\mathcal{L}(\alpha) \cap \mathcal{L}(M)$ ,  $\alpha \in \text{DRX}_{\text{vsf}}$  and  $M \in \text{DFA}$ .*

The unbounded size of  $\Sigma$  comes from the PSPACE-hardness of the intersection emptiness problem for DRX by Martens et al. [31], which has the same requirement. Using the existential theory of concatenation for the upper bound might seem conceptually excessive – but this cannot be avoided (see Section A.18 in the full version of the paper [21]).

We now combine the proofs of Theorems 6 and 29, and observe:

► **Theorem 30.** *Given  $M_1, M_2 \in \text{DTMFA}_{\text{mcf}}$ ,  $\mathcal{L}(M_1) \subseteq \mathcal{L}(M_2)$  can be decided in PSPACE.*

Obviously, this implies that equivalence for  $\text{DTMFA}_{\text{mcf}}$  is decidable in PSPACE, and, furthermore, this also holds for  $\text{DRX}_{\text{vsf}}$ , which is an interesting contrast to non-deterministic  $\text{RX}_{\text{vsf}}$ : As shown by Freydenberger [18], equivalence (and, hence, inclusion and minimization) are undecidable for  $\text{RX}_{\text{vsf}}$  (while [18] does not explicitly mention the concept, the regex in that proof are vstar-free, as discussed in [20]). Hence, Theorem 30 also yields a minimization algorithm for  $\text{DRX}_{\text{vsf}}$  and  $\text{DTMFA}_{\text{mcf}}$  that works in PSPACE (enumerate all smaller candidates and check equivalence). We leave open whether this is optimal, but observe that even for DREG, minimization is NP-complete, see Niewerth [33].

Next, we discuss a potential extension of determinism. One could argue that Definition 14 is overly restrictive; e. g., consider  $\alpha := \langle x: \mathbf{a}^+ \rangle \langle y: \mathbf{b}^+ \rangle c(\&x \vee \&y)$ . Then  $\alpha$  is not deterministic; but as the contents of  $x$  and  $y$  always start with  $\mathbf{a}$  or  $\mathbf{b}$  (respectively), deterministic choices between  $\&x$  and  $\&y$  are possible by looking at the current letter of the input word. Analogous observations can be made for TMFA. More precisely, we define the notion of  $\ell$ -deterministic TMFA as a relaxation of the criteria of DTMFA: In contrast to the latter, an  $\ell$ -deterministic TMFA can have states  $q$  with multiple memory recall-transitions, as long as these recall distinct memories, and if  $q$  is reached in some computation, then for each pair of these recalled memories, the contents differ in the first  $\ell$  positions. First, note that this does not increase the expressive power (intuitively, storing the length  $\ell$  prefixes of the memory contents allows making  $\ell$ -deterministic memory recall transitions deterministic):

► **Proposition 31.** *Let  $\ell \geq 1$ . For every  $\ell$ -deterministic  $M \in \text{DTMFA}$ , there is an  $M' \in \text{DTMFA}$  with  $\mathcal{L}(M) = \mathcal{L}(M')$ .*

For the sake of the argument, let  $\alpha \in \text{RX}$  be  $\ell$ -deterministic if and only if  $\mathcal{M}(\alpha)$  is.

► **Proposition 32.** *For every  $\ell \geq 1$ , deciding whether a TMFA is  $\ell$ -deterministic is PSPACE-complete. The problem is coNP-complete if the input is restricted to  $\text{TMFA}_{\text{mcf}}$ . These lower bounds hold even if we restrict the input to  $\text{RX}$  and  $\text{RX}_{\text{vsf}}$ , respectively.*

Hence, while we can decide efficiently whether a TMFA or a regex is deterministic, detecting  $\ell$ -determinism is costly, even for  $\ell = 1$ . The same holds if we adapt the definition to distinguish between variables and terminals (see Section A.21 in the full version of the paper [21]).



## 7 Conclusions and Further Directions

Based on TMFA, an automaton model for regex, we extended the notion of determinism from regular expressions to regex. Although the resulting language class cannot express all regular languages, it is still rich; and by using a generalization of the Glushkov construction, deterministic regex can be converted into a DTMFA, and the membership problem can then be solved quite efficiently. Although we did not discuss this, the construction is also compatible with the Glushkov construction with counters by Gelade, Gyssens, Martens [23]. Hence, one can add counters to DRX and DTMFA without affecting the complexity of membership.

Many challenging questions remain open, for example: Can the more advanced results for DREG be adapted to DRX, i. e., can  $\mathcal{M}(\alpha)$  be computed more efficiently (as in [8, 34]), or is it even possible, like in [24], to avoid computing  $\mathcal{M}(\alpha)$ ? Is effective minimization possible for DTMFA or DRX? Is it decidable whether a DTMFA defines a DRX-language? Are inclusion and equivalence decidable for DRX or DTMFA? Can determinism be generalized to larger classes of regex without making the membership problem intractable?

**Acknowledgements.** The authors thank Wim Martens for helpful feedback, Matthias Niewerth, for pointing out that  $v_n$  must be a factor of  $p_n$  in the jumping lemma, and Martin Braun, for creating a library and tool for DRX and DTMFA (available at [6]).

---

### References

- 1 Abigail. Re: Random number in perl. Posting in the newsgroup comp.lang.perl.misc, October 1997. Message-ID slrn64sudh.qp.abigail@betelgeuse.wayne.fnx.com.
- 2 Alfred V. Aho. Algorithms for finding patterns in strings. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 5, pages 255–300. Elsevier, Amsterdam, 1990.
- 3 Dana Angluin. Finding patterns common to a set of strings. *J. Comput. Syst. Sci.*, 21:46–62, 1980.
- 4 Pablo Barceló, Carlos A. Hurtado, Leonid Libkin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. In *Proc. PODS 2010*, 2010.
- 5 Geert Jan Bex, Wouter Gelade, Frank Neven, and Stijn Vansummeren. Learning deterministic regular expressions for the inference of schemas from XML data. *ACM Trans. Web*, 4(4):14, 2010.
- 6 Martin Braun. moar – Deterministic Regular Expressions with Backreferences, 2016. Accessed December 2016. URL: <https://github.com/s4ke/moar>.
- 7 Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible markup language XML 1.0 (fifth edition). W3C recommendation. Technical Report <https://www.w3.org/TR/2008/REC-xml-20081126/>, W3C, November 2008.
- 8 Anne Brüggemann-Klein. Regular expressions into finite automata. *Theor. Comput. Sci.*, 120(2):197–213, 1993.
- 9 Anne Brüggemann-Klein and Derick Wood. One-unambiguous regular languages. *Inf. Comput.*, 142(2):182–206, 1998.
- 10 Cezar Câmpeanu, Kai Salomaa, and Sheng Yu. A formal study of practical regular expressions. *Int. J. Found. Comput. Sci.*, 14:1007–1018, 2003.
- 11 Benjamin Carle and Paliath Narendran. On extended regular expressions. In *Proc. LATA 2009*, 2009.
- 12 Wojciech Czerwinski, Claire David, Katja Losemann, and Wim Martens. Deciding definability by deterministic regular expressions. In *Proc. FOSSACS 2013*, pages 289–304, 2013.
- 13 Volker Diekert. Makanin’s Algorithm. In *Algebraic Combinatorics on Words* [29], chapter 12.

- 14 Volker Diekert, Artur Jeż, and Wojciech Plandowski. Finding all solutions of equations in free groups and monoids with involution. In *Proc. CSR 2014*, 2014.
- 15 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document spanners: A formal approach to information extraction. *J. ACM*, 62(2):12, 2015.
- 16 Henning Fernau and Markus L. Schmid. Pattern matching with variables: A multivariate complexity analysis. *Inform. Comput.*, 242:287–305, 2015.
- 17 Henning Fernau, Markus L. Schmid, and Yngve Villanger. On the parameterised complexity of string morphism problems. *Theory Comput. Syst.*, 59(1):24–51, 2016.
- 18 Dominik D. Freydenberger. Extended regular expressions: Succinctness and decidability. *Theory Comput. Syst.*, 53(2):159–193, 2013.
- 19 Dominik D. Freydenberger. A logic for document spanners. In *Proc. ICDT 2017*, 2017. Accepted. Available at <http://ddfy.de/publications/F-ALfDS.html>.
- 20 Dominik D. Freydenberger and Mario Holldack. Document spanners: From expressive power to decision problems. In *Proc. ICDT 2016*, 2016.
- 21 Dominik D. Freydenberger and Markus L. Schmid. Deterministic regular expressions with back-references. A version of this paper that also includes the Appendix. URL: <http://ddfy.de/publications/FS-DREwBR.html>.
- 22 Shudi (Sandy) Gao, C. M. Sperberg-McQueen, and Henry S. Thompson. W3C XML schema definition language (XSD) 1.1 part 1: Structures. Technical Report <https://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/>, W3C, April 2012.
- 23 Wouter Gelade, Marc Gyssens, and Wim Martens. Regular expressions with counting: Weak versus strong determinism. *SIAM J. Comput.*, 41(1):160–190, 2012.
- 24 Benoît Groz, Sebastian Maneth, and Slawek Staworko. Deterministic regular expressions in linear time. In *Proc. PODS 2012*, 2012.
- 25 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
- 26 S. C. Kleene. Representation of events in nerve nets and finite automata. In C. E. Shannon, J. McCarthy, and W. R. Ashby, editors, *Automata Studies*, pages 3–42. Princeton University Press, Princeton, NJ, 1956.
- 27 Markus Latte and Matthias Niewerth. Definability by weakly deterministic regular expressions with counters is decidable. In *Proc. MFCS 2015*, 2015.
- 28 Katja Losemann, Wim Martens, and Matthias Niewerth. Closure properties and descriptive complexity of deterministic regular expressions. *Theor. Comput. Sci.*, 627:54–70, 2016.
- 29 M. Lothaire. *Algebraic Combinatorics on Words*, volume 90 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2002.
- 30 Ping Lu, Joachim Bremer, and Haiming Chen. Deciding determinism of regular languages. *Theory Comput. Syst.*, 57(1):97–139, 2015.
- 31 Wim Martens, Frank Neven, and Thomas Schwentick. Complexity of decision problems for XML schemas and chain regular expressions. *SIAM J. Comput.*, 39(4):1486–1530, 2009.
- 32 Makoto Murata, Dongwon Lee, Murali Mani, and Kohsuke Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM TOIT*, 5(4):660–704, 2005.
- 33 Matthias Niewerth. *Data Definition Languages for XML Repository Management Systems*. PhD thesis, TU Dortmund, 2015. URL: [http://www.theoinf.uni-bayreuth.de/en/downloads/PHD\\_Niewerth.pdf](http://www.theoinf.uni-bayreuth.de/en/downloads/PHD_Niewerth.pdf).
- 34 Jean-Luc Ponty, Djelloul Ziadi, and Jean-Marc Champarnaud. A new quadratic algorithm to convert a regular expression into an automaton. In *Proc. WIA '96*, 1996.
- 35 Markus L. Schmid. Characterising REGEX languages by regular languages equipped with factor-referencing. *Inform. Comput.*, 249:1–17, 2016.

# On the Decomposition of Finite-Valued Streaming String Transducers

Paul Gallot<sup>1</sup>, Anca Muscholl<sup>2</sup>, Gabriele Puppis<sup>3</sup>, and Sylvain Salvati<sup>4</sup>

- 1 ENS Paris-Saclay, Paris, France  
pgallot@ens-paris-saclay.fr
- 2 Université de Bordeaux, LaBRI & CNRS, Bordeaux, France  
anca@labri.fr
- 3 Université de Bordeaux, LaBRI & CNRS, Bordeaux, France  
gabriele.puppis@labri.fr
- 4 Université de Lille 1, CRISAL & INRIA, Lille, France  
sylvain.salvati@univ-lille1.fr

---

## Abstract

We prove the following decomposition theorem: every 1-register streaming string transducer that associates a uniformly bounded number of outputs with each input can be effectively decomposed as a finite union of functional 1-register streaming string transducers. This theorem relies on a combinatorial result by Kortelainen concerning word equations with iterated factors. Our result implies the decidability of the equivalence problem for the considered class of transducers. This can be seen as a first step towards proving a more general decomposition theorem for streaming string transducers with multiple registers.

**1998 ACM Subject Classification** F.1.1 Models of Computation, Automata

**Keywords and phrases** Streaming Transducers, finite valuedness, equivalence

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.34

## 1 Introduction

Data management systems heavily depend upon models of transformation of data that must be efficient both in terms of processing time and in terms of memory resources, and yet enjoy decidable static analysis methods (e.g. algorithms for checking equivalence).

Finite transducers, that is, finite automata extended with outputs, are simple devices that enable some effective, and even efficient reasoning on data transformations. These models come in different variants, depending on whether they use non-determinism or not, on whether the input is scanned only once from left to right (one-way transducers) or several times and in different directions (two-way transducers, or two-way generalised sequential machines 2GSM), and on the number of outputs that can be associated with each input. Concerning the last property, transducers that associate at most one output with each input are called functional transducers.

One may expect that the transformations computed by finite state transducers can be equally described in logic, like monadic second-order logic (MSO), but this holds only up to a certain extent. For instance, two-way transducers are as expressive as MSO-definable transductions [6] when restricted to the functional case, but they become incomparable as soon as unrestricted (i.e. relational) transductions are considered.



© Paul Gallot, Anca Muscholl, Gabriele Puppis, and Sylvain Salvati;  
licensed under Creative Commons License CC-BY

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 34; pp. 34:1–34:14

Leibniz International Proceedings in Informatics



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

Streaming String Transducers (SSTs) [2] have been introduced as a mechanical representation of transducers described by means of MSO. They gained a growing interest in a series of works [1, 2, 3, 4], where their formal properties have been investigated and where they have been put to applications to model transformations on data streams, heaps etc. Intuitively, SSTs are one-way automata enhanced with a finite number of registers to store and manipulate partial outputs. The registers can be updated by appending and prepending symbols, and possibly by concatenating several registers into one; the register content, however, cannot be inspected to control the flow of the computation.

Contrary to two-way transducers, not only functional SSTs are as expressive as MSO-definable functional transductions, but also this remains true when considering relational transductions. Therefore, as for MSO transductions, functional SSTs are equally expressive as functional two-way transducers and incomparable to them for relational transductions. Somehow, the equivalence of SSTs with MSO transductions is rather intuitive in view of a result [8] that characterizes MSO tree transductions in terms of finite copying macro tree transducers with the single use restriction. In SSTs, the finite copying mechanism is represented by the use of a finite number of registers, while the single use restriction for SSTs is formalized as the requirement that, during a transition of the machine, the content of each register can be used at most once to produce the new contents (*copyless* SSTs). Moreover, this result relates SSTs to much older formalisms, notably the Lindenmayer's L systems called D0L and HDT0L, which can be used to describe transductions and whose generalizations to trees can be seen, in a certain way, as macro tree transducers.

This paper focuses on finite-valued transductions, which lie strictly between functional and relational ones. Formally, a transduction is finite-valued if it associates a uniformly bounded number of outputs (say, at most  $k$  outputs) with each input. For one-way transducers it is known how to decide whether it is finite-valued [17], and in this case compute the maximal number of outputs associated with some input [11]. While the decidability of this property is still an open question for two-way transducers. What is more interesting is that, even if finite-valued transducers capture more transductions than functional transducers, they often enjoy properties similar to the latter ones, especially concerning the decidability of the fundamental problems and the relationship with logic.

Of course, transducers beyond the classical one-way case pose new challenges with respect to the fundamental problems, notably, the equivalence problem. For example, equivalence is known to be decidable for functional SSTs, even without the copyless restriction [9], but it is undecidable for unrestricted (i.e. relational) ones [10]. On the other hand, the equivalence problem remains decidable for finite-valued 2GSMs [12], by an argument relying on Ehrenfeuchts's conjecture. The latter paper actually shows that every HDT0L language  $L$  has an effective test set, that is, a finite subset on which equivalence of finite-valued 2GSM w.r.t.  $L$  can be tested.

The pervasive similarity between finite-valued and functional transducers can sometimes be explained by a quite strong *decomposition theorem* of the following form, where  $\mathcal{C}$  is an appropriate class of transducers:

*Within  $\mathcal{C}$  finite-valued transducers are equivalent to finite unions of functional transducers.*

It is known from [18] that the above statement holds when  $\mathcal{C}$  is the class of one-way transducers. Our main interest is in proving the decomposition theorem for the class of SSTs, since, as we will see, this implies solutions to a certain number of open problems, notably, the decidability of the equivalence problem for finite-valued SSTs and the equivalence to finite-valued two-way transducers. We begin this investigation by proving a decomposition theorem for the sub-class

of SSTs with 1 register. Unfortunately, we are not able to generalize this result to SSTs with multiple registers. Towards the end of the paper we outline a few points of our proof that are problematic for the generalization.

## 2 Preliminaries

**Two-way transducers.** A *two-way transducer*, or *two-way generalized sequential machine* (abbreviated *2GSM*), is a tuple  $\mathcal{T} = (\Sigma, \Gamma, Q, I, E, F)$ , where  $\Sigma$  (resp.  $\Gamma$ ) is a finite input (resp. output) alphabet,  $Q$  is a finite set of states,  $I$  (resp.  $F$ ) is a set of initial (resp. final) states included in  $Q$ , and  $E \subseteq Q \times \Sigma \times \Gamma^* \times Q \times \{-1, +1\}$  is a finite set of transition rules describing, for each state and input symbol, the possible output string, target state, and direction of movement. To correctly define the transitions at the extremities of the input, we use two special symbols  $\triangleright$  and  $\triangleleft$  and assume that the input of a 2GSM is of the form  $w = a_1 \dots a_n$ , with  $n \geq 2$ ,  $a_1 = \triangleright$ ,  $a_n = \triangleleft$ , and  $a_i \neq \triangleright, \triangleleft$  for all  $i = 2, \dots, n-1$ . We say that  $\mathcal{T}$  is *one-way* if it can only move to the right, that is,  $(q, a, v, q', d) \in E$  implies  $d = +1$ .

A *configuration* of  $\mathcal{T}$  on input  $w$  is a pair  $(q, p) \in Q \times \{1, \dots, |w|\}$  consisting of a state and a position in the input. The *transitions* of  $\mathcal{T}$  on  $w$  connect pairs of configurations and are of the form  $(q, p) \xrightarrow{a/v} (q', p')$ , with  $a = w(p)$ ,  $(q, a, v, q', d) \in E$ , and  $p' = p + d$ . We assume that  $\mathcal{T}$  can only move right (resp. left) at the first (resp. last) position of  $w$ , that is,  $(q, \triangleright, v, q', d) \in E$  implies  $d = +1$  and  $(q, \triangleleft, v, q', d) \in E$  implies  $d = -1$ . A *run* of  $\mathcal{T}$  on  $w$  is any sequence of transitions of the form  $(q_0, p_0) \xrightarrow{w(p_0)/v_1} (q_1, p_1) \xrightarrow{w(p_1)/v_2} \dots \xrightarrow{w(p_{n-1})/v_n} (q_n, p_n)$ . The *output* of a run  $\rho$  is the word  $\text{out}(\rho) = v_1 v_2 \dots v_n$ . The run  $\rho$  is *initial* (resp. *final*) if  $q_0 \in I$  and  $p_0 = 1$  (resp. if  $q_n \in F$  and  $p_n = |w|$ ).

**Streaming string transducers.** For convenience, we define streaming string transducers in a slightly different but equivalent way compared to [2] (this concerns in particular the definition of the output). A *streaming string transducer* (SST) is a tuple  $\mathcal{T} = (\Sigma, \Gamma, Q, X, x_{\text{out}}, U, I, E, F)$ , where  $\Sigma, \Gamma$  are the usual input and output alphabets,  $Q$  is the state space,  $X$  is a finite set of registers disjoint from  $\Gamma$ ,  $x_{\text{out}} \in X$  is a special register used to describe the final output,  $U$  is a finite set of *register updates*, namely, functions from  $X$  to  $(X \uplus \Gamma)^*$ ,  $I$  is a subset of  $Q$  representing the initial states,  $E \subseteq Q \times \Sigma \times U \times Q$  is a set of transition rules, describing, for each state and input symbol, the possible updates and target states, and  $F \subseteq Q$  is a set of final states.

To define the semantics of  $\mathcal{T}$ , we make use of *valuations*, that is, functions  $g : X \rightarrow \Gamma^*$ . A valuation  $g$  can be homomorphically extended to words over  $X \uplus \Gamma$  and to register updates, as follows. Given a word  $w \in (X \uplus \Gamma)^*$ ,  $g(w)$  is the word over  $\Gamma$  obtained from  $w$  by replacing every occurrence of a register  $x$  with its valuation  $g(x)$ . Similarly, given an update  $f : X \rightarrow (X \uplus \Gamma)^*$ ,  $g \circ f$  is the valuation that maps each register  $x$  to the word  $g(f(x))$ . A *configuration* of  $\mathcal{T}$  is a pair state-valuation  $(q, g)$ . The configuration is *initial* (resp. *final*) if  $q \in I$  and  $g(x) = \varepsilon$  for all registers  $x \in X$  (resp.  $q \in F$ ). When reading a symbol  $a$ , the SST moves from configuration  $(q, g)$  to configuration  $(q', g')$  if there is a transition rule  $(q, a, f, q') \in E$  such that  $g' = g \circ f$ . We denote such a transition by  $(q, g) \xrightarrow{a/f} (q', g')$ . A *run* of  $\mathcal{T}$  on  $w = a_1 \dots a_n$  is any sequence of transitions of the form  $(q_0, g_0) \xrightarrow{a_1/f_1} (q_1, g_1) \xrightarrow{a_2/f_2} \dots \xrightarrow{a_n/f_n} (q_n, g_n)$ . It is *initial* (resp. *successful*) if it begins with an initial configuration (resp. if it begins with an initial configuration and ends with a final configuration). The *output* of a successful run is the last valuation  $g_n(x_{\text{out}})$  of the special register  $x_{\text{out}}$ .

A well-behaved class of SSTs is obtained by requiring that all the updates are copyless. Formally, an SST  $\mathcal{T} = (\Sigma, \Gamma, Q, X, x_{\text{out}}, U, I, E, F)$  is *copyless* if for every update  $f \in U$ , every

register  $x \in X$  appears at most once in  $f(x_1) \cdot \dots \cdot f(x_k)$ , where  $X = \{x_1, \dots, x_k\}$ . Hereafter we tacitly assume that all SSTs are copyless.

**Unambiguity, transductions, functionality.** The following definitions apply to both two-way transducers and streaming string transducers. The *input automaton* of a transducer  $\mathcal{T}$  is a finite automaton  $\mathcal{A}$  obtained from  $\mathcal{T}$  by removing any information concerned with the construction of the output (e.g. the output alphabet, the registers, etc.). For example, if  $\mathcal{T}$  is an SST, then the input automaton  $\mathcal{A}$  of  $\mathcal{T}$  has transition rules of the form  $(q, a, q')$  for all transition rules  $(q, a, f, q')$  of  $\mathcal{T}$ . This automaton recognizes precisely the language of input words  $w$  that have at least one successful run of  $\mathcal{T}$  (and hence at least one associated output). We say that  $\mathcal{T}$  is *input-unambiguous* (resp. *input- $k$ -ambiguous*) if its input automaton is unambiguous, namely, it admits at most one successful run on each input (resp. if the input automaton admits at most  $k$  successful runs on each input).

A *transduction* is a relation  $T \subseteq \Sigma^* \times \Gamma^*$  consisting of pairs of input and output words. It is *computed* by a transducer  $\mathcal{T}$  if it contains exactly those pairs  $(w, v) \in \Sigma^* \times \Gamma^*$  for which there is a successful run of  $\mathcal{T}$  on  $w$  with output  $v$ .  $\mathcal{T}$  is called *functional* if it computes a transduction which is a partial function, namely, it associates at most one output with each input. Similarly, it is  *$k$ -valued* if it associates at most  $k$  outputs with each input (in [4] such a transducer is called *finitary*).

Transductions can be also defined in monadic second-order logic (*MSO*). Intuitively, this is done by specifying an output (seen as a relational structure) from a fixed number of copies of the input. We only give a short account of the definitions and we refer the reader to [5] for the details. An *MSO transduction with  $m$  copies* consists of a formula  $\Phi_{\text{dom}}(\bar{Z})$ , where  $\bar{Z} = (Z_1, \dots, Z_m)$  is an  $m$ -tuple of free monadic variables, a formula  $\Phi_a^i(\bar{Z}, x)$  for each  $i \in \{1, \dots, m\}$  and  $a \in \Gamma$ , and a formula  $\Phi_{\leq}^{i,j}(\bar{Z}, x, y)$  for each  $i, j \in \{1, \dots, m\}$ . For different valuations of the free variables  $\bar{Z}$ , which range over sets of positions of the input, we may have different outputs associated with that input. The formula  $\Phi_{\text{dom}}(\bar{Z})$  describes which valuations generate an output. The formula  $\Phi_a^i(\bar{Z}, x)$  tells whether, in the output corresponding to the valuation of  $\bar{Z}$ , the position  $x$  in the  $i$ -th copy of the input is an element of the output and, in this case, if it is labeled with the letter  $a$ . Similarly, the formula  $\Phi_{\leq}^{i,j}(x, y)$  tells whether, in the output corresponding to the valuation of  $\bar{Z}$ , the element  $x$  of the  $i$ -th copy of the input precedes the element  $y$  of the  $j$ -th copy of the input.

### 3 The conjectured decomposition theorem and its implications

We conjecture the following decomposition theorem for SSTs:

► **Conjecture 1.**  *$k$ -valued SSTs are effectively equivalent to finite unions of functional SSTs.*

In Section 5 we will give a proof of this conjecture in the restricted case of SSTs with a single register. Even if we are not able to prove the result in its full generality, we discuss here some important implications of the conjecture, which motivate our interest in having such a result. A first consequence would be the decidability of equivalence for  $k$ -valued SSTs:

► **Corollary 2.** *Assuming that Conjecture 1 holds, one can decide if two  $k$ -valued SSTs  $\mathcal{T}, \mathcal{T}'$  are equivalent.*

**Proof.** Conjecture 1 implies that  $\mathcal{T} = \mathcal{T}_1 \cup \dots \cup \mathcal{T}_k$  and  $\mathcal{T}' = \mathcal{T}'_1 \cup \dots \cup \mathcal{T}'_k$  where  $\mathcal{T}_1, \dots, \mathcal{T}_k, \mathcal{T}'_1, \dots, \mathcal{T}'_k$  are all functional transducers. From [4], we know that functional SSTs can be made deterministic, and also that the inclusion (as a transduction) of an SST within a finite

union of deterministic SSTs is decidable. A consequence is that we can decide whether both  $\mathcal{T} \subseteq \mathcal{T}'_1 \cup \dots \cup \mathcal{T}'_k$  and  $\mathcal{T}' \subseteq \mathcal{T}_1 \cup \dots \cup \mathcal{T}_k$  hold, that is, whether  $\mathcal{T}$  is equivalent to  $\mathcal{T}'$ . ◀

Another implication concerns the relationship between SSTs and 2GSMs:

► **Corollary 3.** *Assuming that Conjecture 1 holds,  $k$ -valued SSTs and  $k$ -valued 2GSMs are equally expressive. In addition, this would give a decomposition theorem for  $k$ -valued 2GSMs.*

**Proof.** From [2] we know that functional SSTs are expressively equivalent to functional MSO-definable transductions, and from [7] the latter are expressively equivalent to functional 2GSMs. So functional SSTs can be translated to equivalent functional 2GSMs. Our conjecture would then imply that  $k$ -valued SSTs can be transformed to equivalent 2GSMs.

The proof that  $k$ -valued 2GSMs can be translated to equivalent SSTs is sketched below. Note that this translation does not rely on any decomposition theorem for 2GSMs. In fact, the previous arguments paired with the following proof, do imply a decomposition theorem for  $k$ -valued 2GSMs. Consider a 2GSM  $\mathcal{T}$  and define the *crossing number* of a successful run  $\rho$  of  $\mathcal{T}$  as the maximum number of configurations that occur in  $\rho$  at same position  $p$  of the underlying input. Using classical techniques from [16], one can encode with a regular language any set of successful runs of  $\mathcal{T}$  with uniformly bounded crossing numbers. Using such an encoding and a number of registers proportional to the bound on the crossing numbers, one can then simulate the output produced by those runs in a single left-to-right pass, that is by means of an SST  $\mathcal{T}'$ . In general, the transduction computed by  $\mathcal{T}'$  is included in that of  $\mathcal{T}$ . The crucial observation is that, if  $\mathcal{T}$  is  $k$ -valued, then all its outputs can be produced by successful runs that have crossing number at most  $|Q|$ , where  $Q$  is the state space of  $\mathcal{T}$ . The SST  $\mathcal{T}'$  is thus equivalent to  $\mathcal{T}$ . ◀

## 4 The classical decomposition theorem for one-way transducers

We give a short overview of the proof of a classical decomposition theorem for one-way transducers, following the presentation of [15]:

► **Theorem 4** ([18]).  *$k$ -valued one-way transducers are effectively equivalent to finite unions of functional one-way transducers.*

We present this proof, not only for self-containment, but also because it allows us to isolate the key important properties that need to be generalized in order to obtain a decomposition theorem for SSTs. After the proof sketch, we will mention some important implications of the theorem, notably, the decidability of the equivalence problem and the correspondence with a suitable subclass of MSO-definable transductions.

Given two automata or transducers  $\mathcal{A}, \mathcal{B}$ , we say that  $\mathcal{B}$  is a *covering* of  $\mathcal{A}$  if there is a function  $h$  from the states of  $\mathcal{B}$  to the states of  $\mathcal{A}$  such that

- (i)  $h$  preserves the transitions, that is, if  $\tau = (q, a, \dots, q')$  is a transition rule of  $\mathcal{B}$ , then  $h(\tau) \stackrel{\text{def}}{=} (h(q), a, \dots, h(q'))$  is a transition rule of  $\mathcal{A}$ ,
- (ii)  $h$  induces a bijection between the initial states of  $\mathcal{B}$  and the initial states of  $\mathcal{A}$ ,
- (iii) for all states  $q$  of  $\mathcal{B}$ ,  $h$  induces a bijection between the transitions of  $\mathcal{B}$  exiting  $q$  and the transitions of  $\mathcal{A}$  exiting  $h(q)$ , and
- (iv) both  $h$  and  $h^{-1}$  preserve final states, that is,  $q$  is final in  $\mathcal{A}$  iff  $h(q)$  is final in  $\mathcal{B}$ .

Note that when  $\mathcal{B}$  is a covering of  $\mathcal{A}$ , there is a bijection between the (successful) runs of  $\mathcal{B}$  and the (successful) runs of  $\mathcal{A}$ ; in particular,  $\mathcal{B}$  and  $\mathcal{A}$  are equivalent (they recognize the same language or compute the same transduction). A common way to define a covering of  $\mathcal{A}$  consists in equipping the states of  $\mathcal{A}$  with additional information which can be maintained

in a deterministic way during the transitions. In this case, the states of the covering can be denoted by pairs of the form  $(q, o)$ , where  $q$  is a state of  $\mathcal{A}$  and  $o$  is some object (the additional information) associated with  $q$ . Accordingly, given a run  $\rho$  of such a covering of  $\mathcal{A}$ , we denote by  $\rho|_1$  the corresponding run of  $\mathcal{A}$ .

Let  $\mathcal{T}$  be a one-way transducer and  $\rho, \rho'$  two runs of it on the same input and with possibly different outputs  $\text{out}(\rho), \text{out}(\rho')$ . We would like to give a “measure” of the divergence of these outputs. The size of a pair of words  $(u, v) \in \Gamma^* \times \Gamma^*$  is the sum of the lengths of  $u$  and  $v$ , i.e.  $|(u, v)| = |u| + |v|$ . The *lead or delay* of a pair of words  $(x, y)$ , denoted  $\text{LD}(x, y)$  is either the smallest pair of words  $(u, v)$  such that  $xu = yv$ , if such a pair exists, or  $\perp$  otherwise. Note that if  $\text{LD}(\text{out}(\rho), \text{out}(\rho')) \neq \perp$ , then this is a pair where at least one of the two components is the empty word  $\varepsilon$ , and it is  $(\varepsilon, \varepsilon)$  iff  $\rho$  and  $\rho'$  produce the same output. We also define the *lag* of  $(\rho, \rho')$  as the number

$$\Delta(\rho, \rho') \stackrel{\text{def}}{=} \max_{\rho = \alpha\beta, \rho' = \alpha'\beta', |\alpha| = |\alpha'|} |\text{LD}(\text{out}(\alpha), \text{out}(\alpha'))|$$

(by convention, we let  $|\perp| = \infty > n$  for all  $n \in \mathbb{N}$ ). Intuitively,  $\Delta(\rho, \rho')$  describes how much the partial outputs produced by some prefixes of  $\rho$  and  $\rho'$  of the same length have diverged.

The proof of the decomposition theorem consists in arguing as follows.

**The main combinatorial property.** Given a  $k$ -valued one-way transducer  $\mathcal{T}$ , one constructs a multi-transducer  $\mathcal{T}^{k+1}$ , whose successful runs can be seen as  $(k+1)$ -tuples of successful runs of  $\mathcal{T}$  spelling out the same input and possibly different outputs. One proves the following combinatorial property (see [18]): *for every successful run  $\bar{\rho} = (\rho_1, \dots, \rho_{k+1})$  of  $\mathcal{T}^{k+1}$ , there exist distinct components  $\rho_i, \rho_j$  of  $\bar{\rho}$  that not only agree on the output (this is necessary since  $\mathcal{T}$  is  $k$ -valued), but also have  $\Delta(\rho_i, \rho_j) \leq n_0$ , where  $n_0$  is a large enough constant computed from  $\mathcal{T}^{k+1}$ .*

The idea underlying this result is that if two runs  $\rho, \rho'$  of  $\mathcal{T}$  on the same input have produced at some point partial outputs that are sufficiently far from each other, i.e.  $\Delta(\alpha, \alpha') > n_0$  for some prefixes  $\alpha, \alpha'$  of  $\rho, \rho'$ , then this difference can be augmented so much that it cannot be overtaken by the suffixes of  $\rho, \rho'$  that come after  $\alpha, \alpha'$  — this gives two modified runs that produce different outputs. As  $\mathcal{T}$  is  $k$ -valued, if one considers  $k+1$  pairs of runs, then at least two of them must produce the same output and have lag at most  $n_0$ .

**The lag separation covering.** Let  $Q$  be the state space of  $\mathcal{T}$  and  $<$  a lexicographical order (induced by a total order on transitions) between pairs of runs of  $\mathcal{T}$  with the same input, i.e.  $\rho < \rho'$  implies  $\rho, \rho'$  have the same input. One can construct a covering  $\mathcal{U}_{n_0}$  of  $\mathcal{T}$ , whose states are the pairs  $(q, o)$ , with  $q \in Q$  and  $o : Q \rightarrow 2^{\Gamma^* \times \Gamma^*}$ , such that for all initial runs  $\rho$  of  $\mathcal{U}_{n_0}$  ending in  $(q, o)$  and all states  $r \in Q$ ,

$$\begin{aligned} \text{out}(\rho) &= \text{out}(\rho|_1) && \text{(namely, the output produced by the run } \rho \text{ of } \mathcal{U}_{n_0} \\ & && \text{is the same as that of the underlying run } \rho|_1 \text{ of } \mathcal{T}) \\ o(r) &= \left\{ \text{LD}(\text{out}(\rho|_1), \text{out}(\rho')) : \begin{array}{l} \rho' \text{ initial run of } \mathcal{T} \text{ ending in } r \\ \text{such that } \rho' < \rho|_1 \text{ and } \Delta(\rho|_1, \rho) \leq n_0 \end{array} \right\} \end{aligned}$$

(note that the above pairs  $\text{LD}(\text{out}(\rho|_1), \text{out}(\rho'))$  have size at most  $n_0$ , so the transducer  $\mathcal{U}_{n_0}$  is finite; for a proof of how these pairs can be maintained in the transitions of  $\mathcal{U}_{n_0}$  see [15]).

From  $\mathcal{U}_{n_0}$ , one can then construct a transducer  $\mathcal{V}_{n_0}$  whose successful runs with input  $w$  and output  $x$  simulate precisely the *least* runs among the successful ones of  $\mathcal{T}$  with input  $w$  and output  $x$ . Formally,  $\mathcal{V}_{n_0}$  is obtained from  $\mathcal{U}_{n_0}$  by restricting the set of final states to those pairs  $(q, o)$  such that  $q$  is final in  $\mathcal{T}$  and  $(\varepsilon, \varepsilon) \notin o(r)$  for all  $r \in Q$ . Indeed, we have:



1. If  $\rho$  is a successful run of  $\mathcal{V}_{n_0}$ , then  $\rho|_1$  is a successful run of  $\mathcal{T}$  and for all other successful runs  $\rho'$  of  $\mathcal{T}$  with the same input and the same output, either  $\rho|_1 < \rho'$  or  $\Delta(\rho|_1, \rho) > n_0$ .
2. For all pairs  $(w, x)$  in the transduction defined by  $\mathcal{T}$ , if  $\rho'$  is the *least* successful run of  $\mathcal{T}$  with input  $w$  and output  $x$ , then there is a successful run  $\rho$  of  $\mathcal{V}_{n_0}$  such that  $\rho' = \rho|_1$ .

This shows that  $\mathcal{V}_{n_0}$  is equivalent to  $\mathcal{T}$ .

In addition, the transducer  $\mathcal{V}_{n_0}$  is *input- $k$ -ambiguous*. This relies on the choice of  $n_0$  and can be shown by way of contradiction as follows. Suppose that  $\mathcal{V}_{n_0}$  admits  $k + 1$  distinct successful runs  $\rho_1, \dots, \rho_{k+1}$  on the same input  $w$ . By the combinatorial property stated above, there exist two indices  $i \neq j$  such that  $\rho_i|_1$  and  $\rho_j|_1$  have the same input, the same output, and satisfy  $\Delta(\rho_i|_1, \rho_j|_1) \leq n_0$ . However, this would contradict property 1. above.

**The multi-skimming covering.** The last part of the proof is fully generic and consists in decomposing a  $k$ -ambiguous automaton  $\mathcal{A}$  (e.g. the input automaton of  $\mathcal{V}_{n_0}$ ) into a finite union of unambiguous automata  $\mathcal{A}_1, \dots, \mathcal{A}_k$  (the construction can be easily lifted to the transducer  $\mathcal{V}_{n_0}$  by copying the outputs generated by the transitions).

Let  $\mathcal{A}$  be a  $k$ -ambiguous automaton with state space  $Q$ . One first defines a covering  $\mathcal{B}$  of  $\mathcal{A}$  whose states are of the form  $(q, o)$ , with  $q \in Q$  and  $o : Q \rightarrow \{0, \dots, k - 1\}$ , such that for all initial runs of  $\mathcal{B}$  ending in  $(q, o)$  and all states  $r$ ,

$$o(r) = |\{\rho' : \rho' \text{ initial run of } \mathcal{A} \text{ ending in } r \text{ such that } \rho' < \rho|_1\}|$$

(see again [15] for a construction that maintains the function  $o$  along the transitions of  $\mathcal{B}$ ).

Finally, one constructs some automata  $\mathcal{B}_0, \dots, \mathcal{B}_{k-1}$  that differ from  $\mathcal{B}$  only in the sets of final states. Formally, for each  $i = 0, \dots, k - 1$ , one declares a state  $(q, o)$  final in  $\mathcal{B}_i$  iff  $q \in F$  and  $\sum_{r \in F} o(r) = i$ , where  $F$  is the set of final states of  $\mathcal{A}$ . Note that an initial run  $\rho$  of the automaton  $\mathcal{B}_i$  is successful iff  $\rho|_1$  is successful for  $\mathcal{A}$  and there are exactly  $i$  successful runs of  $\mathcal{A}$  that precede  $\rho|_1$  in the lexicographic ordering. In particular, the automata  $\mathcal{B}_0, \dots, \mathcal{B}_{k-1}$  are all unambiguous and their union is equivalent to  $\mathcal{A}$ .

## 5 The proof of the conjecture for 1-register SSTs

In this section we prove a decomposition theorem for finite-valued SSTs with a single register:

► **Theorem 5.**  *$k$ -valued 1-register SSTs are effectively equivalent to finite unions of functional 1-register SSTs.*

From the sketch of the proof of the decomposition theorem for one-way transducers we see that there are two important results that need to be generalized in order to lift the theorem to SSTs. The first of them is the combinatorial property related to the lag  $\Delta(\rho, \rho')$ . A noticeable difference is that, in the presence of an SST  $\mathcal{T}$  with 1 register, partial outputs are constructed by adding symbols both to the right and to the left of the register of  $\mathcal{T}$ . This means that, to correctly measure the distance between the register contents  $x, x'$  of runs, we need to take into account the possible contexts that can be added to both endpoints of  $x$  and  $x'$ . We thus modify the definition of *lead or delay* of a pair  $(x, x')$ :

$$\text{LD}(x, x') \stackrel{\text{def}}{=} \{(u, v, u', v') : uxv = u'x'v', (u = \varepsilon) \vee (u' = \varepsilon), (v = \varepsilon) \vee (v' = \varepsilon)\}.$$

The size of a tuple  $\lambda = (u, v, u', v') \in \text{LD}(x, x')$  is  $|\lambda| = |u| + |u'| + |v| + |v'|$ .

Let  $\text{reg}(\rho)$  denote the register content at the end of an initial run  $\rho$  of  $\mathcal{T}$ . Note that this is precisely the output produced by  $\rho$  if  $\rho$  is successful. Intuitively, a tuple  $\lambda \in \text{LD}(\text{reg}(\rho), \text{reg}(\rho'))$

describes a possible alignment between the register contents at the end of  $\rho$  and  $\rho'$ . Such a tuple has one of the following forms (the annotations under the tuples describe succinctly the type of alignment between  $\text{reg}(\rho)$  and  $\text{reg}(\rho')$ ):



Accordingly, we define the *lag* of a pair of initial runs  $(\rho, \rho')$  as the number

$$\Delta(\rho, \rho') \stackrel{\text{def}}{=} \max_{\rho = \alpha \beta, \rho' = \alpha' \beta', |\alpha| = |\alpha'|} \min_{\lambda \in \text{LD}(\text{reg}(\alpha), \text{reg}(\alpha'))} |\lambda|$$

(as usual, we assume that  $\min \emptyset = \infty > n$  for all  $n \in \mathbb{N}$ ).

**The combinatorial property.** Along the same line of the proof in Section 4, given a  $k$ -valued SST  $\mathcal{T}$  with 1 register, we construct the SST  $\mathcal{T}^{k+1}$  with  $k+1$  registers that simulates  $k+1$  copies of  $\mathcal{T}$  running in parallel on the same input. Given a run  $\bar{\rho} = (\rho_1, \dots, \rho_{k+1})$  of  $\mathcal{T}^{k+1}$ , we denote by  $\bar{\rho}|_i$  the  $i$ -th run  $\rho_i$  of  $\bar{\rho}$ . To correctly work with  $\mathcal{T}^{k+1}$ , we need to assume that  $\mathcal{T}$  has been normalized in such a way that the register content can only grow. Formally, we say that  $\mathcal{T} = (\Sigma, \Gamma, Q, \{x\}, x, U, I, E, F)$  is *normalized* if for all register updates  $f \in U$ ,  $f(x)$  contains exactly one occurrence of  $x$ . It is easy to transform any 1-register SST into an equivalent normalized SST.

► **Lemma 6.** *One can normalize any 1-register SST.*

We now turn towards the main combinatorial property:

► **Proposition 7.** *Given a normalized  $k$ -valued SST  $\mathcal{T}$ , one can compute  $n_0 \in \mathbb{N}$  such that, for all successful runs  $\bar{\rho}$  of  $\mathcal{T}^{k+1}$ ,  $\text{reg}(\rho|_i) = \text{reg}(\rho|_j)$  and  $\Delta(\rho|_i, \rho|_j) \leq n_0$  for some  $i \neq j$ .*

Recall that the technique from [18] for proving the combinatorial result for one-way transducers consists in augmenting the differences between pairs of runs of  $\bar{\rho}$ , until the runs are shown to produce pairwise distinct outputs. In the case of SSTs, it becomes difficult to apply such a technique. Instead, we rely on a powerful result of Kortelainen [13] which has been later improved and simplified by Saarela (Theorem 9 below) and that is related to solutions of word equations with iterated factors. Word equations will indeed be obtained by comparing the outputs produced by pumped versions of successful runs of  $\mathcal{T}$ . Formally, we call *loop* of a run  $\rho$  any factor of  $\rho$  that starts and ends in the same state. A loop of  $\rho$  is *maximal* if it is not strictly contained in any other loop of  $\rho$ . Given a set  $L$  of pairwise non-overlapping loops of  $\rho$ , we denote by  $\text{pump}_L^h(\rho)$  the run obtained from  $\rho$  by repeating  $h$  times the loops in the set  $L$  (if  $h = 0$  this amounts at removing the loops). The following lemma is simple but crucial for our result (notice that it does not hold in general for SSTs with more than one register).

► **Lemma 8.** *Given an initial run  $\rho$  of a normalized 1-register SST and a set  $L$  of pairwise non-overlapping loops of  $\rho$ , one can factorize  $\text{reg}(\rho)$  as  $u_0 v_1 u_1 \dots u_{2m-1} v_{2m} u_{2m}$  in such a way that, for all  $h \in \mathbb{N}$ ,  $\text{reg}(\text{pump}_L^h(\rho)) = u_0 (v_1)^h u_1 \dots u_{2m-1} (v_{2m})^h u_{2m}$ .*

► **Theorem 9** (Theorem 4.3 in [14]). *The set of solutions of an equation of the form*

$$u_0 (v_1)^h u_1 \dots u_{m-1} (v_m)^h u_m = u'_0 (v'_1)^h u'_1 \dots u'_{m-1} (v'_m)^h u'_m$$

where  $h$  is the unknown, is either  $\mathbb{N}$  or a finite subset of  $\mathbb{N}$ .

Since the loops of any run  $\bar{\rho}$  of  $\mathcal{T}^{k+1}$  can be equally seen as loops of the runs  $\rho|_1, \dots, \rho|_{k+1}$  of  $\mathcal{T}$ , we can write  $\text{pump}_L^h(\rho|_i)$  whenever  $L$  is a set of pairwise non-overlapping loops of  $\bar{\rho}$ . The following property is an immediate consequence of Lemma 8 and Theorem 9:

► **Corollary 10.** *For every successful run  $\bar{\rho}$  of  $\mathcal{T}^{k+1}$ , every set  $L$  of pairwise non-overlapping loops of  $\bar{\rho}$ , and every pair of indices  $i, j \in \{1, \dots, k+1\}$ ,*

$$\text{reg}(\rho|_i) \neq \text{reg}(\rho|_j) \quad \text{implies} \quad \exists h_0 \in \mathbb{N} \quad \forall h \geq h_0 \quad \text{reg}(\text{pump}_L^h(\rho|_i)) \neq \text{reg}(\text{pump}_L^h(\rho|_j)).$$

We are now ready to prove Proposition 7 by way of contradiction, that is, we fix a large enough  $n_0$  (see Lemma 11 below for the precise value of  $n_0$ ) and we assume that there is a successful run  $\bar{\rho}$  of  $\mathcal{T}^{k+1}$  such that, for all  $i, j$ ,  $\text{reg}(\rho|_i) = \text{reg}(\rho|_j)$  implies  $\Delta(\rho|_i, \rho|_j) > n_0$ , and we show that  $\mathcal{T}$  outputs at least  $k+1$  different words on the same input  $w$ , a contradiction with the  $k$ -valuedness of  $\mathcal{T}$ . We will do so by exploiting an induction on the number of pairs of runs  $\rho|_i, \rho|_j$  with the same output. For the sake of brevity, we denote by  $c_{\mathcal{T}}$  the *capacity* of  $\mathcal{T}$ , that is, the maximum number of symbols that are added to the content of the register during a single transition of  $\mathcal{T}$ . We further define  $E(\bar{\rho}) = \{(i, j) \in \{1, \dots, k+1\} : \text{reg}(\rho|_i) = \text{reg}(\rho|_j)\}$ . We prove the following invariant, which leads towards a contradiction of  $k$ -valuedness of  $\mathcal{T}$ :

► **Lemma 11.** *If there is a successful run  $\bar{\rho}$  of  $\mathcal{T}^{k+1}$  with  $E(\bar{\rho}) \neq \emptyset$  and*

$$\forall i, j \quad (i, j) \in E(\bar{\rho}) \quad \text{implies} \quad \Delta(\rho|_i, \rho|_j) > c_{\mathcal{T}} \cdot |Q|^{k+1} \quad (*)$$

then there is another such run  $\bar{\rho}'$  of  $\mathcal{T}^{k+1}$  satisfying  $E(\bar{\rho}') \not\subseteq E(\bar{\rho})$  and  $(*)$ .

**Proof.** For every pair  $(i, j) \in E(\bar{\rho})$ , we can split the run  $\bar{\rho}$  at the first point where the lag exceeds the value  $c_{\mathcal{T}} \cdot |Q|^{k+1}$ , that is, we can factorize  $\bar{\rho}$  as  $\bar{\alpha}_{i,j} \bar{\alpha}'_{i,j}$  in such a way that  $\bar{\alpha}_{i,j}$  is the shortest prefix satisfying  $\Delta(\bar{\alpha}_{i,j}|_i, \bar{\alpha}_{i,j}|_j) > c_{\mathcal{T}} \cdot |Q|^{k+1}$ . Based on this, we choose a pair  $(i, j) \in E(\bar{\rho})$  that induces a factorization of  $\bar{\rho}$  with the longest prefix  $\bar{\alpha}_{i,j}$ . Clearly, for all other pairs  $(i', j') \in E(\bar{\rho})$ ,  $\bar{\alpha}_{i',j'}$  is a prefix of  $\bar{\alpha}_{i,j}$ , and hence

$$\Delta(\bar{\alpha}_{i,j}|_{i'}, \bar{\alpha}_{i,j}|_{j'}) \geq \Delta(\bar{\alpha}_{i',j'}|_{i'}, \bar{\alpha}_{i',j'}|_{j'}) = \Delta(\bar{\rho}|_{i'}, \bar{\rho}|_{j'}) > c_{\mathcal{T}} \cdot |Q|^{k+1}$$

Now that the pair  $(i, j)$  is fixed, we further factorize the suffix  $\bar{\alpha}'_{i,j}$  of  $\bar{\rho}$  as follows:

$$\bar{\alpha}'_{i,j} = \bar{\beta}_0 \bar{\gamma}_1 \bar{\beta}_1 \dots \bar{\beta}_{m-1} \bar{\gamma}_m \bar{\beta}_m$$

where  $L = \{\bar{\gamma}_1, \dots, \bar{\gamma}_m\}$  is a maximal set of pairwise non-overlapping maximal loops of  $\bar{\alpha}'_{i,j}$ . Further let  $\bar{\rho}^{(h)} = \text{pump}_L^h(\bar{\rho})$ , for all  $h \in \mathbb{N}$ , and observe that  $\bar{\rho}^{(1)} = \bar{\rho}$ .

We claim that there exists  $h_{i,j} \in \mathbb{N}$  such that for all  $h \geq h_{i,j}$ ,  $\text{reg}(\bar{\rho}^{(h)}|_i) \neq \text{reg}(\bar{\rho}^{(h)}|_j)$ . Indeed, if this were not the case, then there would exist infinitely many  $h \in \mathbb{N}$  such that  $\text{reg}(\bar{\rho}^{(h)}|_i) = \text{reg}(\bar{\rho}^{(h)}|_j)$ . In particular, by Theorem 9, we would have  $\text{reg}(\bar{\rho}^{(h)}|_i) = \text{reg}(\bar{\rho}^{(h)}|_j)$  for  $h = 0$ . Note that  $\bar{\alpha}_{i,j}$  is also prefix of  $\bar{\rho}^{(0)}$  and the remaining part of  $\bar{\rho}^{(0)}$  has length at most  $|Q|^{k+1}$ , since otherwise it would contain a loop which is either contained in one of the  $\bar{\beta}_i$ 's, in which case we would get a contradiction with the maximality of  $L$ , or is spread between several  $\bar{\beta}_i$ 's, which would then be in contradiction with the maximality of the loops  $\bar{\gamma}_i$ . Now, consider the register contents at the end of the  $i$ -th and  $j$ -th runs of  $\bar{\alpha}_{i,j}$  and  $\bar{\rho}^{(0)}$ ; that is,

let  $x = \text{reg}(\bar{\alpha}_{i,j}|_i)$ ,  $x' = \text{reg}(\bar{\alpha}_{i,j}|_j)$ ,  $x_0 = \text{reg}(\bar{\rho}^{(0)}|_i)$ ,  $x'_0 = \text{reg}(\bar{\rho}^{(0)}|_j)$ . Since  $\mathcal{T}$  is normalized, there exist some words  $u, v, u', v'$  such that  $x_0 = u x v$  and  $x'_0 = u' x' v'$ . Moreover, since these words represent the content that is added to the registers by the updates performed along a suffix of  $\bar{\rho}^{(0)}$  of length at most  $|Q|^{k+1}$ , we know that  $u v$  and  $u' v'$  have length at most  $c_{\mathcal{T}} \cdot |Q|^{k+1}$ . Since  $x_0 = \text{reg}(\bar{\rho}^{(0)}|_i) = \text{reg}(\bar{\rho}^{(h)}|_i) = x'_0$ , we know that  $(u, v, u', v') \in \text{LD}(x, x')$ . But, by definition,  $\Delta(\bar{\alpha}_{i,j}|_i, \bar{\alpha}_{i,j}|_j) = \min\{|\lambda| : \lambda \in \text{LD}(x, x')\} \leq c_{\mathcal{T}} \cdot |Q|^{k+1}$  and therefore,  $\Delta(\bar{\alpha}_{i,j}|_i, \bar{\alpha}_{i,j}|_j) \leq |u| + |v| + |u'| + |v'| \leq c_{\mathcal{T}} \cdot |Q|^{k+1}$  which contradicts the definition of  $\bar{\alpha}_{i,j}$ .

We have just proved that there is  $h_{i,j} \in \mathbb{N}$  such that, for all  $h \geq h_{i,j}$ ,  $(i, j) \notin E(\bar{\rho}^{(h)})$ . Consider now the other pairs  $(i', j') \in \{1, \dots, k+1\}^2 \setminus E(\bar{\rho})$ , namely, such that  $\text{reg}(\bar{\rho}|_{i'}) \neq \text{reg}(\bar{\rho}|_{j'})$ . By Corollary 10, for every such pair  $(i', j')$  there is  $h_{i',j'} \in \mathbb{N}$  such that, for all  $h \geq h_{i',j'}$ ,  $\text{reg}(\bar{\rho}^{(h)}|_{i'}) \neq \text{reg}(\bar{\rho}^{(h)}|_{j'})$ . We can thus define  $\bar{\rho}' = \bar{\rho}^{(h')}$ , where  $h'$  is the maximum of the  $h_{i',j'}$ 's for all  $(i', j') \notin E(\bar{\rho})$ . In this way we get  $(i, j) \notin E(\bar{\rho}')$  and  $E(\bar{\rho}') \subsetneq E(\bar{\rho})$ . Finally, it remains to verify that  $\bar{\rho}'$  satisfies the condition  $(\star)$ . Consider a pair  $(i', j') \in E(\bar{\rho}')$ . Since  $\bar{\alpha}_{i',j'}$  is a prefix of  $\bar{\rho}'$  and  $\Delta(\bar{\alpha}_{i',j'}|_{i'}, \bar{\alpha}_{i',j'}|_{j'}) > c_{\mathcal{T}} \cdot |Q|^{k+1}$ , we conclude that  $\Delta(\bar{\rho}'|_{i'}, \bar{\rho}'|_{j'}) > c_{\mathcal{T}} \cdot |Q|^{k+1}$  as well.  $\blacktriangleleft$

**The lag separation covering.** The second and last point that we need to explain is the construction of a lag separation covering of  $\mathcal{T}$ , that is, a 1-register SST  $\mathcal{U}_{n_0}$  with the following properties. The states of  $\mathcal{U}_{n_0}$  are the pairs  $(q, o)$ , with  $q \in Q$ ,  $Q$  state space of  $\mathcal{T}$ , and  $o$  some stored information determining a certain function  $\hat{o} : Q \rightarrow 2^{\Gamma^* \times \Gamma^* \times \Gamma^* \times \Gamma^*}$  (we will see soon why  $o$  cannot be directly defined as the function  $\hat{o}$  itself). For all initial runs  $\rho$  of  $\mathcal{U}_{n_0}$  ending in  $(q, o)$  and all states  $r \in Q$ , we must have

$$\begin{aligned} \text{reg}(\rho) &= \text{reg}(\rho|_1) && \text{(namely, the register content at the end of the run } \rho \text{ of } \mathcal{U}_{n_0} \\ &&& \text{is the same as that of the underlying run } \rho|_1 \text{ of } \mathcal{T}) \\ \hat{o}(r) &= \bigcup \left\{ \text{LD}(\text{reg}(\rho|_1), \text{reg}(\rho')) : \begin{array}{l} \rho' \text{ initial run of } \mathcal{T} \text{ ending in } r \\ \text{such that } \rho' < \rho|_1 \text{ and } \Delta(\rho|_1, \rho) \leq n_0 \end{array} \right\}. \end{aligned}$$

One can then transform  $\mathcal{U}_{n_0}$  into an input- $k$ -ambiguous SST  $\mathcal{V}_{n_0}$  equivalent to  $\mathcal{T}$ : this is done as explained in Section 4 by restricting the set of final states to those pairs  $(q, o)$  such that  $q$  is final in  $\mathcal{T}$  and  $(\varepsilon, \varepsilon, \varepsilon, \varepsilon) \notin \hat{o}(r)$  for all  $r \in Q$ . The proof that  $\mathcal{V}_{n_0}$  is input- $k$ -ambiguous relies on the combinatorial property for normalized 1-register SSTs that we gave before. Finally, one applies the multi-skimming covering construction to  $\mathcal{V}_{n_0}$  in order to obtain  $k$  input-unambiguous SSTs  $\mathcal{T}_0, \dots, \mathcal{T}_{k-1}$  whose union is equivalent to  $\mathcal{T}$ .

For the sake of brevity, let  $\Lambda^{\leq n_0}$  be the set of tuples  $\lambda = (u, v, u', v')$  of size at most  $n_0$ , and let  $\text{LD}^{\leq n_0}(x, x') = \text{LD}(x, x') \cap \Lambda^{\leq n_0}$ . The construction of  $\mathcal{U}_{n_0}$  boils down to showing that one can maintain the information about the set  $\text{LD}^{\leq n_0}(\text{reg}(\rho), \text{reg}(\rho'))$  while processing any two runs  $\rho, \rho'$  of  $\mathcal{T}$ , provided that  $\Delta(\rho, \rho') \leq n_0$ . Indeed, suppose that, by some means, we are able to construct a deterministic automaton  $\mathcal{A}$ , with state space  $S$ , together with a function  $\hat{\cdot} : S \rightarrow 2^{\Lambda^{\leq n_0}}$  such that, whenever  $\mathcal{A}$  reads an initial run  $(\rho, \rho')$  of  $\mathcal{T}^2$  (seen as a sequence of pairs of transition rules of  $\mathcal{T}$ ), then  $\mathcal{A}$  reaches a state  $s_{\rho, \rho'}$  such that  $\hat{s}_{\rho, \rho'}$  is either the set  $\text{LD}^{\leq n_0}(\text{reg}(\rho), \text{reg}(\rho'))$  or the empty set, depending on whether  $\Delta(\rho, \rho') \leq n_0$  or not. Using the automaton  $\mathcal{A}$  one can construct the covering  $\mathcal{U}_{n_0}$  as follows:

- The states of  $\mathcal{U}_{n_0}$  are the pairs  $(q, o)$ , with  $q \in Q$  and  $o \in S \times (2^S)^Q$ . The component  $q$  is for simulating an initial run  $\rho$  of  $\mathcal{T}$  on the given input. The stored information  $o$  consists of two parts: a state  $s \in S$  for simulating the run of  $\mathcal{A}$  on  $(\rho, \rho)$  and a function  $t : Q \rightarrow 2^S$  for simulating the possible runs of  $\mathcal{A}$  on inputs of the form  $(\rho, \rho')$ , for all initial runs  $\rho'$  of  $\mathcal{T}$  such that  $\rho' < \rho$ , with a partitioning based on the last state of  $\rho'$ . The latter part  $t$  also determines the function  $\hat{o} : Q \rightarrow 2^{\Lambda^{\leq n_0}}$  associated with the stored information, that is,  $\hat{o}(r) = \bigcup_{s \in t(r)} \hat{s}$  for all  $r \in Q$ .

- The possible transitions of  $\mathcal{U}_{n_0}$  are of the form  $(q_1, o_1) \xrightarrow[\mathcal{U}_{n_0}]{a/f} (q_2, o_2)$ , with  $o_1 = (s_1, t_1)$  and  $o_2 = (s_2, t_2)$ , whenever  $\tau = (q_1, a, f, q_2)$  is a transition rule of  $\mathcal{T}$ ,  $(s_1, (\tau, \tau), s_2)$  is a transition rule of  $\mathcal{A}$ , and, for all  $r_2 \in Q$ ,

$$t_2(r_2) \stackrel{\text{def}}{=} \left\{ s' : \begin{array}{l} (s_1, (\tau, \tau'), s') \text{ transition rule of } \mathcal{A}, \\ \tau' = (q_1, a, f', r_2) \text{ transition rule of } \mathcal{T}, \quad \tau' < \tau \end{array} \right\} \\ \cup \left\{ s' : \begin{array}{l} s \in t_1(r_1), \quad (s, (\tau, \tau'), s') \text{ transition rule of } \mathcal{A}, \\ r_1 \in Q, \quad \tau' = (r_1, a, f', r_2) \text{ transition rule of } \mathcal{T} \end{array} \right\}.$$

- The initial states of  $\mathcal{U}_{n_0}$  are the pairs  $(q, o_0)$ , with  $o_0 = (s_0, t_0)$ ,  $q$  initial state of  $\mathcal{T}$ ,  $s_0$  initial state of  $\mathcal{A}$ , and  $t_0(r) = \emptyset$  for all  $r \in Q$ . Similarly, the final states of  $\mathcal{U}_{n_0}$  are the pairs  $(q, o)$ , with  $q$  final state of  $\mathcal{T}$ .

**Maintaining alignments under updates.** After the previous argument, we are left to outline the construction of an automaton  $\mathcal{A}$  that recognizes pairs of runs  $(\rho, \rho')$  and from the states of which we can extract  $\text{LD}^{\leq n}(\text{reg}(\rho), \text{reg}(\rho'))$  when  $\Delta(\rho, \rho') \leq n$ , for some fixed number  $n$ . More specifically, we can focus on the following sub-goal: for a fixed  $n \in \mathbb{N}$ , maintain a suitable data structure in bounded memory (the set of these structures form the state space of  $\mathcal{A}$ ) that determines  $\text{LD}^{\leq n}(x, x')$  while the registers  $x, x'$  are being updated, as usual under the proviso that this set remains non-empty all along the computation. A possible way to attain this sub-goal is to try to compute the set  $\text{LD}^{\leq n}(sxt, s'x't')$  directly from a given set  $\text{LD}^{\leq n}(x, x')$  and some given words  $s, t, s', t'$ . Note that the actual content of the registers  $x, x'$  is not part of the input of this computational problem. Along this idea, we disclose the dependencies of the sets  $\text{LD}^{\leq n}(xa, x')$  and  $\text{LD}^{\leq n}(ax, x')$  from  $\text{LD}^{\leq n+1}(x, x')$ :

$$\text{LD}^{\leq n}(xa, x') = \left\{ (u, \varepsilon, u', \mathbf{v}'\mathbf{a}) : (u, \varepsilon, u', \mathbf{v}') \in \text{LD}^{\leq n-1}(x, x') \right\} \\ \cup \left\{ (u, \mathbf{v}, u', \varepsilon) : (u, \mathbf{a}\mathbf{v}, u', \varepsilon) \in \text{LD}^{\leq n+1}(x, x') \right\} \quad (1)$$

$$\text{LD}^{\leq n}(ax, x') = \left\{ (\varepsilon, v, \mathbf{a}\mathbf{u}', v') : (\varepsilon, v, \mathbf{u}', v') \in \text{LD}^{\leq n-1}(x, x') \right\} \\ \cup \left\{ (\mathbf{u}, v, \varepsilon, v') : (\mathbf{u}\mathbf{a}, v, \varepsilon, v') \in \text{LD}^{\leq n+1}(x, x') \right\}. \quad (2)$$

Using these dependencies, one can describe any set of the form  $\text{LD}^{\leq n}(\tilde{x}, \tilde{x}')$ , where  $\tilde{x} = sxt$  and  $\tilde{x}' = s'x't'$ , on the basis of the set  $\text{LD}^{\leq \tilde{n}}(x, x')$  and the words  $s, t, s', t'$ , where  $\tilde{n} = n + |s| + |t| + |s'| + |t'|$ . However, the fact that  $\tilde{n}$  may be larger than  $n$  seems to require a knowledge of alignments of arbitrarily large size, which we cannot assume. Luckily we can overcome this problem by exploiting the condition that  $\text{LD}^{\leq n}(x, x') \neq \emptyset$  and by using an auxiliary data structure for maintaining some knowledge about the periodicities in  $x$  and  $x'$ .

We say that a word  $x$  has period  $k \in \mathbb{N}$  if  $x(i) = x(i+k)$  for all  $1 \leq i \leq |x| - k$ . For example,  $abcab$  has period 3, but also period 5, 6, 7, etc. We define the two sets

$$R(x) \stackrel{\text{def}}{=} \left\{ (u, p) : x = uy \text{ and } yp \text{ is periodic with period } |p| \right\} \quad \begin{array}{c} \overbrace{u} \quad \overbrace{p} \quad \overbrace{p} \\ \hline x \end{array} \\ L(x) \stackrel{\text{def}}{=} \left\{ (v, q) : x = yv \text{ and } yq \text{ is periodic with period } |q| \right\} \quad \begin{array}{c} \overbrace{q} \quad \overbrace{q} \quad \overbrace{v} \\ \hline x \end{array}$$

Intuitively,  $(u, p) \in R(x)$  iff  $x$  begins with  $u$  and continues with a word  $y$  whose period  $|p|$  is preserved under extensions with  $p$  to the right, and similarly for  $(v, q) \in L(x)$  (see the figure). As usual, we let  $R^{\leq m}(x)$  (resp.  $L^{\leq m}(x)$ ) be the restriction of  $R(x)$  (resp.  $L(x)$ ) to the pairs of size at most  $m$  (the size being the sum of the lengths of the words in the pair).

For any fixed  $m \in \mathbb{N}$ , it is possible to devise a bounded-memory data structure that can be maintained while the register  $x$  is being updated and that, at every moment, determines, the sets  $R^{\leq m}(x)$  and  $L^{\leq m}(x)$ . The data structure basically recalls the following information:

- the exact content of  $x$  when its length is smaller than  $m$ ,
  - the unique pair of words  $(u, p) \in R^{\leq m}(x)$  (if it exists) that has the smallest length  $|u| + |p|$ ,
  - the unique pair of words  $(v, q) \in L^{\leq m}(x)$  (if it exists) that has the smallest length  $|v| + |q|$ .
- From now on we assume that the sets  $R^{\leq m}(x)$ ,  $L^{\leq m}(x)$ ,  $R^{\leq m}(x')$ ,  $L^{\leq m}(x')$  are known during any sequence of updates of  $x$  and  $x'$ .

Below we show that any non-empty set  $\text{LD}^{\leq n}(x, x')$ , paired with the information given by the sets  $R^{\leq \tilde{n}}(x)$ ,  $L^{\leq \tilde{n}}(x)$ ,  $R^{\leq \tilde{n}}(x')$ ,  $L^{\leq \tilde{n}}(x')$ , fully determines the content of  $\text{LD}^{\leq \tilde{n}}(x, x')$ . Thanks to the dependencies given in Equations (1) and (2), this is sufficient for computing also the set  $\text{LD}^{\leq n}(\tilde{x}, \tilde{x}')$ , where  $\tilde{x}$  (resp.  $\tilde{x}'$ ) is obtained from  $x$  (resp.  $x'$ ) by prepending/appending at most  $\tilde{n} - n$  symbols.

We define two sets,  $G_{n,m}(x, x')$  and  $H_{n,m}(x, x')$ , on the basis of the sets  $\text{LD}^{\leq n}(x, x')$ ,  $L^{\leq m}(x)$ ,  $R^{\leq m}(x)$ ,  $L^{\leq m}(x')$ ,  $R^{\leq m}(x')$ . The elements of  $G_{n,m}(x, x')$  represent possible tuples of alignment type  $\bar{-}$ :

$$\begin{aligned}
 G_{n,m}(x, x') = & \left\{ (\varepsilon, \mathbf{p}\mathbf{v}, \mathbf{u}'\mathbf{q}, \varepsilon) : (\varepsilon, \mathbf{v}, \mathbf{u}', \varepsilon) \in \text{LD}^{\leq n}(x, x'), \begin{array}{l} (\mathbf{u}', \mathbf{p}) \in P^{\leq m}(x), \\ (\mathbf{v}, \mathbf{q}) \in Q^{\leq m}(x'), \end{array} |p| = |q| \right\} \\
 \cup & \left\{ (\varepsilon, \mathbf{v}, \mathbf{u}', \varepsilon) : (u, \varepsilon, \varepsilon, v') \in \text{LD}^{\leq n}(x, x'), \begin{array}{l} (\varepsilon, p) \in P^{\leq m}(x), \\ (\varepsilon, q) \in Q^{\leq m}(x'), \end{array} p = \mathbf{v}v', \quad |p| = |q| \right\} \\
 \cup & \left\{ (\varepsilon, \mathbf{v}, \mathbf{u}'\mathbf{q}, \varepsilon) : (\varepsilon, \varepsilon, \mathbf{u}', v') \in \text{LD}^{\leq n}(x, x'), \begin{array}{l} (\mathbf{u}', p) \in P^{\leq m}(x), \\ (\varepsilon, \mathbf{q}) \in Q^{\leq m}(x'), \end{array} p = \mathbf{v}v', \quad |p| = |q| \right\} \\
 \cup & \left\{ (\varepsilon, \mathbf{p}\mathbf{v}, \mathbf{u}', \varepsilon) : (u, \mathbf{v}, \varepsilon, \varepsilon) \in \text{LD}^{\leq n}(x, x'), \begin{array}{l} (\varepsilon, \mathbf{p}) \in P^{\leq m}(x), \\ (\mathbf{v}, \mathbf{q}) \in Q^{\leq m}(x'), \end{array} q = \mathbf{u}\mathbf{u}', \quad |p| = |q| \right\}.
 \end{aligned}$$

The set  $H_{n,m}(x, x')$  is defined in a symmetric way, as if the underlying order of the words were reversed. Accordingly, its elements represent possible tuples of alignment type  $\bar{-}$ . For the sake of brevity, we omit the tedious definition of  $H_{n,m}(x, x')$ .

The following lemma gives a way to compute the set  $\text{LD}^{\leq \tilde{n}}(x, x')$  (and hence  $\text{LD}^{\leq n}(\tilde{x}, \tilde{x}')$ ) for any  $\tilde{x} = sxt$  and  $\tilde{x}' = s'x't'$ , with  $|s| + |t| + |s'| + |t'| \leq \tilde{n} - n$ : for this it suffices to construct  $\text{LD}^{\leq n}(x, x') \cup G_{n, n+\tilde{n}}(x, x') \cup H_{n, n+\tilde{n}}(x, x')$  and then remove from it the tuples of size larger than  $\tilde{n}$ . Thanks to the lemma, this results in the set  $\text{LD}^{\leq \tilde{n}}(x, x')$ .

► **Lemma 12.** *For all  $n \leq \tilde{n}$ , if  $\text{LD}^{\leq n}(x, x')$  is non-empty, then*

$$\text{LD}^{\leq \tilde{n}}(x, x') \subseteq \text{LD}^{\leq n}(x, x') \cup G_{n, n+\tilde{n}}(x, x') \cup H_{n, n+\tilde{n}}(x, x') \subseteq \text{LD}(x, x').$$

Tuples  $(\text{LD}^{\leq n}(x, x'), L^{\leq 2n+2c\tau}(x), R^{\leq 2n+2c\tau}(x), L^{\leq 2n+2c\tau}(x'), R^{\leq 2n+2c\tau}(x'))$  thus form an adequate state space for  $\mathcal{A}$  when  $x$  and  $x'$  are the contents of the registers of the runs  $\mathcal{A}$  is recognizing. This completes the proof of Theorem 5. And, as for Corollary 2, we obtain:

► **Corollary 13.** *One can decide if two  $k$ -valued 1-register SSTs  $\mathcal{T}, \mathcal{T}'$  are equivalent.*

## 6 Conclusions

We have proven a decomposition theorem for finite-valued 1-register SSTs, generalizing a similar result of Weber for one-way transducers. This result constitutes a milestone towards a decomposition theorem for finite-valued SSTs with multiple registers paving the way to the decidability of the equivalence problem, and to a correspondence with finite-valued two-way transducers. This way meets however two difficulties.

The first difficulty is related to establishing the analogous of Proposition 7. Indeed, the latter proposition relies on the exact correspondence between the number of iterations of

loops in a successful run of an SST and the number of iterations of certain factors in the produced output (cf. Lemma 8). This correspondence is broken in the presence of updates that concatenate different registers, namely, functions  $f$  such that, for some register  $x$ ,  $f(x)$  contains at least two occurrences of distinct registers. In particular, iterating a loop may not result in iterating factors in the output and, even worse, it can be the case that iterating loops make two different runs become equal for all iteration except a finite number of them. This is a situation which does not show up when we only deal with one register. In view of this, one is either likely to need a more complicated induction in order to prove the analogous of Proposition 7 for SSTs with multiple registers, or has to pertain to an argument about string combinatorics that is more general than that of Kortelainen and Sarelaa.

Another difficulty, which is perhaps even more challenging, lies in the construction of the lag separation covering of  $\mathcal{T}$ , and more precisely, in the problem of defining an appropriate notion of *lead or delay* between tuples of registers, and in maintaining this object in bounded memory while the registers are being updated. In particular, it seems that taking the obvious generalization of tuple measure we have taken for the 1-register case cannot be maintained using a finite memory.

---

## References

- 1 R. Alur. Streaming string transducers. In *Proceedings of the 18th International Workshop on Logic, Language, Information and Computation (WoLLIC)*, volume 6642 of *LNCS*, page 1, 2011.
- 2 R. Alur and P. Cerny. Expressiveness of streaming string transducers. In *Proceedings of the 36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 1–12, 2010.
- 3 R. Alur and P. Cerny. Streaming transducers for algorithmic verification of single-pass list processing programs. In *Proceedings of the 38th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 2011.
- 4 R. Alur and J. V. Deshmukh. Nondeterministic streaming state transducers. In *Proceedings of the 38th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 6756 of *LNCS*, pages 1–20, 2011.
- 5 B. Courcelle. Monadic second-order graph transductions: a survey. *Theoretical Computer Science*, 126:53–75, 1994.
- 6 J. Engelfriet and H. Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic*, 2(2):216–254, 2001.
- 7 J. Engelfriet and H. J. Hoogeboom. MSO-definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic*, 2(2):216–254, 2001.
- 8 J. Engelfriet and S. Maneth. Macro Tree Transducers, Attribute Grammars, and MSO Definable Tree Translations. *Information and Computation*, 154(1):34–91, 1999.
- 9 E. Filiot and P.-A. Reynier. On streaming string transducers and HDTOL systems. *CoRR*, abs/1412.0537, 2014.
- 10 T. V. Griffiths. The unsolvability of the equivalence problem for lambda-free nondeterministic generalized machines. *Journal of the ACM*, 15(3):409–413, 1968.
- 11 E. M. Gurari and O. H. Ibarra. A note on finite-valued and finitely ambiguous transducers. *Mathematical Systems Theory*, 16(1):61–66, 1983.
- 12 K. Culik II and J. Karhumäki. The equivalence of finite valued transducers (on hdt0l languages) is decidable. *Theoretical Computer Science*, 47:71–84, 1986.
- 13 J. Kortelainen. On the system of word equations  $x_0 u_1^i x_1 u_2^i x_2 \dots u_m^i x_m = y_0 v_1^i y_1 v_2^i y_2 \dots v_m^i y_m$  ( $i = 0, 1, 2, \dots$ ) in a free monoid. *Journal of Automata, Languages and Combinatorics*, 3(1):43–57, 1998.

- 14 A. Saarela. Systems of word equations, polynomials and linear algebra: a new approach. *European Journal of Combinatorics*, 47(5):1–14, 2015.
- 15 J. Sakarovitch and R. De Souza. On the decomposition of  $k$ -valued rational relations. In *Proceedings of the 25th International Symposium on Theoretical Aspects of Computer Science (STACS)*, 2008.
- 16 J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200, 1959.
- 17 A. Weber. On the valuedness of finite transducers. *Acta Informatica*, 27(8):749–780, 1990.
- 18 A. Weber. Decomposing a  $k$ -valued transducer into  $k$  unambiguous ones. *Informatique théorique et applications*, 30(5):379–413, 1996.



# Circuit Evaluation for Finite Semirings\*

Moses Ganardi<sup>1</sup>, Danny HucKe<sup>2</sup>, Daniel König<sup>3</sup>, and Markus Lohrey<sup>4</sup>

1 University of Siegen, Siegen, Germany  
ganardi@eti.uni-siegen.de

2 University of Siegen, Siegen, Germany  
hucKe@eti.uni-siegen.de

3 University of Siegen, Siegen, Germany  
koenig@eti.uni-siegen.de

4 University of Siegen, Siegen, Germany  
lohrey@eti.uni-siegen.de

---

## Abstract

The circuit evaluation problem for finite semirings is considered, where semirings are not assumed to have an additive or multiplicative identity. The following dichotomy is shown: If a finite semiring  $R$  (i) has a solvable multiplicative semigroup and (ii) does not contain a subsemiring with an additive identity  $0$  and a multiplicative identity  $1 \neq 0$ , then its circuit evaluation problem is in  $\text{DET} \subseteq \text{NC}^2$ . In all other cases, the circuit evaluation problem is P-complete.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** circuit value problem, finite semirings, circuit complexity

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.35

## 1 Introduction

Circuit evaluation problems are among the most well-studied computational problems in complexity theory. In its most general formulation, one has an algebraic structure  $\mathcal{A} = (D, f_1, \dots, f_k)$ , where the  $f_i$  are mappings  $f_i : D^{n_i} \rightarrow D$ . A circuit over the structure  $\mathcal{A}$  is a directed acyclic graph (dag) where every inner node is labelled with one of the operations  $f_i$  and has exactly  $n_i$  incoming edges that are linearly ordered. The leaf nodes of the dag are labelled with elements of  $D$  (for this, one needs a suitable finite representation of elements from  $D$ ), and there is a distinguished output node. The task is to evaluate this dag in the natural way, and to return the value of the output node.

In his seminal paper [19], Ladner proved that the circuit evaluation problem for the Boolean semiring  $\mathbb{B}_2 = (\{0, 1\}, \vee, \wedge)$  is P-complete. This result marks a cornerstone in the theory of P-completeness [15], and motivated the investigation of circuit evaluation problems for other algebraic structures. A large part of the literature is focused on commutative (possibly infinite) semirings [1, 23, 31] or circuits with certain structural restrictions (e.g. planar circuits [14, 18, 27] or tree-like circuits [9, 24]). In [25], Miller and Teng proved that circuits over any finite semiring can be evaluated with polynomially many processors in time  $O((\log n)(\log dn))$  on a CRCW PRAM, where  $n$  is the size of the circuit and  $d$  is the formal degree of the circuit. The latter is a parameter that can be exponential in the circuit size  $n$ . On the other hand, the authors are not aware of any NC-algorithms for evaluating

---

\* A full version of the paper is available at <http://arxiv.org/abs/1602.04560>.



general (exponential degree) circuits even for finite semirings. The lack of such algorithms is probably due to Ladner's result, which excludes efficient parallel algorithms in the presence of a Boolean subsemiring unless  $P = NC$ . On the other hand, in the context of semigroups, there exist  $NC$ -algorithms for circuit evaluation. In [8], the following dichotomy result was shown for finite semigroups: If the finite semigroup is solvable (meaning that every subgroup is a solvable group), then circuit evaluation is in  $NC$  (in fact, in  $DET$ , which is the class of all problems that are  $AC^0$ -reducible to the computation of an integer determinant [10, 11]), otherwise circuit evaluation is  $P$ -complete.

In this paper, we extend the work of [8] from finite semigroups to finite semirings. On first sight, Ladner's result seems to exclude efficient parallel algorithms: It is not hard to show that if the finite semiring has an additive identity  $0$  and a multiplicative identity  $1 \neq 0$  (where  $0$  is not necessarily absorbing with respect to multiplication), then circuit evaluation is  $P$ -complete, see Lemma 6. Therefore, we take the most general reasonable definition of semirings: A semiring is a structure  $(R, +, \cdot)$ , where  $(R, +)$  is a commutative semigroup,  $(R, \cdot)$  is a semigroup, and  $\cdot$  distributes (on the left and right) over  $+$ . In particular, we neither require the existence of a  $0$  nor a  $1$ . Our main result states that in this general setting there are only two obstacles to circuit evaluation in  $NC$ : non-solvability of the multiplicative structure and the existence of a zero and a one (different from the zero) in a subsemiring. More precisely, we show the following two results, where a semiring is called  $\{0, 1\}$ -free if there exists no subsemiring with an additive identity  $0$  and a multiplicative identity  $1 \neq 0$ :

1. If a finite semiring is not  $\{0, 1\}$ -free, then the circuit evaluation problem is  $P$ -complete.
2. If a finite semiring  $(R, +, \cdot)$  is  $\{0, 1\}$ -free, then its circuit evaluation problem can be solved with  $AC^0$ -circuits equipped with oracle gates for (a) graph reachability and (b) the circuit evaluation problems for the commutative semigroup  $(R, +)$  and the semigroup  $(R, \cdot)$ .

Together with the dichotomy result from [8] (and the fact that commutative semigroups are solvable) we get the following result: For every finite semiring  $(R, +, \cdot)$ , the circuit evaluation problem is in  $NC$  (in fact, in  $DET$ ) if  $(R, \cdot)$  is solvable and  $(R, +, \cdot)$  is  $\{0, 1\}$ -free. Moreover, if one of these conditions fails, then circuit evaluation is  $P$ -complete.

The hard part of the proof is to show the above statement 2. We will proceed in two steps. In the first step we reduce the circuit evaluation problem for a finite semiring  $R$  to the evaluation of a so-called type admitting circuit. This is a circuit where every gate evaluates to an element of the form  $eah$ , where  $e$  and  $h$  are multiplicative idempotents of  $R$ . Moreover, these idempotents  $e$  and  $h$  have to satisfy a certain compatibility condition that will be expressed by a so-called type function. In a second step, we present a parallel evaluation algorithm for type admitting circuits. Only for this second step we need the assumption that the semiring is  $\{0, 1\}$ -free.

In Section 6 we present an application of our main result for circuit evaluation to formal language theory. We consider the intersection non-emptiness problem for a given context-free language and a fixed regular language  $L$ . If the context-free language is given by an arbitrary context-free grammar, then we show that the intersection non-emptiness problem is  $P$ -complete as long as  $L$  is not empty (Theorem 19). It turns out that the reason for this is non-productivity of nonterminals. We therefore consider a restricted version of the intersection non-emptiness problem, where every nonterminal of the input context-free grammar must be productive. To avoid a promise problem (testing productivity of a nonterminal is  $P$ -complete), we in addition provide a witness of productivity for every nonterminal. This witness consists of exactly one production  $A \rightarrow w$  for every nonterminal of  $A$  where  $w$  may contain nonterminal symbols such that the set of all selected productions is an acyclic grammar  $\mathcal{H}$ . This ensures that  $\mathcal{H}$  derives for every nonterminal  $A$  exactly one string that is a witness of the productivity of  $A$ . We then show that this restricted version of

the intersection non-emptiness problem with the fixed regular language  $L$  is equivalent (with respect to constant depth reductions) to the circuit evaluation problem for a certain finite semiring that is derived from the syntactic monoid of the regular language  $L$ .

Full proofs can be found in the long version [12].

**Further related work.** We mentioned already existing work on circuit evaluation for (possibly infinite) semirings [1, 23, 25, 31]. For infinite groups, the circuit evaluation problem is also known as the compressed word problem [20]. In the context of parallel algorithms, the third and fourth author recently proved that the circuit evaluation problem for finitely generated (but infinite) nilpotent groups belongs to DET [17]. For finite non-associative groupoids, the complexity of circuit evaluation was studied in [26], and some of the results from [8] for semigroups were generalized to the non-associative setting. In [6], the problem of evaluating tensor circuits is studied. The complexity of this problem is quite high: Whether a given tensor circuit over the Boolean semiring evaluates to the  $(1 \times 1)$ -matrix  $(0)$  is complete for nondeterministic exponential time. Finally, let us mention the papers [22, 30], where circuit evaluation problems are studied for the power set structures  $(2^{\mathbb{N}}, +, \cdot, \cup, \cap, \neg)$  and  $(2^{\mathbb{Z}}, +, \cdot, \cup, \cap, \neg)$ , where  $+$  and  $\cdot$  are evaluated on sets via  $A \circ B = \{a \circ b \mid a \in A, b \in B\}$ . Completeness results for a large range of complexity classes are shown in [22, 30].

A variant of our intersection non-emptiness problem was studied in [29]. There, a context-free language  $L$  is fixed, a non-deterministic finite automaton  $\mathcal{A}$  is the input, and the question is, whether  $L \cap L(\mathcal{A}) = \emptyset$  holds. The authors present large classes of context-free languages such that for each member the intersection non-emptiness problem with a given regular language is P-complete (resp., NL-complete).

## 2 Computational complexity

For background in complexity theory the reader might consult [4]. We assume that the reader is familiar with the complexity classes NL (non-deterministic logspace) and P (deterministic polynomial time). A function is logspace-computable if it can be computed by a deterministic Turing-machine with a logspace-bounded work tape, a read-only input tape, and a write-only output tape. Note that the logarithmic space bound only applies to the work tape. P-hardness will refer to logspace reductions.

We use standard definitions concerning circuit complexity, see e.g. [33]. All circuit families in this paper are implicitly assumed to be DLOGTIME-uniform. We will consider the class  $AC^0$  of all problems that can be recognized by a polynomial size circuit family of constant depth built up from NOT-gates (which have fan-in one) and AND- and OR-gates of unbounded fan-in. The class  $NC^k$  ( $k \geq 1$ ) is defined by polynomial size circuit families of depth  $O(\log^k n)$  that use NOT-gates, and AND- and OR-gates of fan-in two. One defines  $NC = \bigcup_{k \geq 1} NC^k$ . The above language classes can be easily generalized to classes of functions by allowing circuits with several output gates. Of course, this only allows to compute functions  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  such that  $|f(x)| = |f(y)|$  whenever  $|x| = |y|$ . If this condition is not satisfied, one has to consider a suitably padded version of  $f$ .

We use the standard notion of constant depth reducibility: For functions  $f_1, \dots, f_k$  let  $AC^0(f_1, \dots, f_k)$  be the class of all functions that can be computed with a polynomial size circuit family of constant depth that uses NOT-gates and unbounded fan-in AND-gates, OR-gates, and  $f_i$ -oracle gates ( $1 \leq i \leq k$ ). Here, an  $f_i$ -oracle gate receives an ordered tuple of inputs  $x_1, x_2, \dots, x_n$  and outputs the bits of  $f_i(x_1 x_2 \dots x_n)$ . By taking the characteristic function of a language, we can also allow a language  $L_i \subseteq \{0, 1\}^*$  in place of  $f_i$ . Note that

the function class  $\text{AC}^0(f_1, \dots, f_k)$  is closed under composition (since the composition of two  $\text{AC}^0$ -circuits is again an  $\text{AC}^0$ -circuit). We write  $\text{AC}^0(\text{NL}, f_1, \dots, f_k)$  for  $\text{AC}^0(\text{GAP}, f_1, \dots, f_k)$ , where GAP is the NL-complete graph accessibility problem. The class  $\text{AC}^0(\text{NL})$  is studied in [3]. It has several alternative characterizations and can be viewed as a nondeterministic version of functional logspace. As remarked in [3], the restriction of  $\text{AC}^0(\text{NL})$  to 0-1 functions is NL. Clearly, every logspace-computable function belongs to  $\text{AC}^0(\text{NL})$ : The NL-oracle can be used to directly compute the output bits of a logspace-computable function.

Let  $\text{DET} = \text{AC}^0(\text{det})$ , where  $\text{det}$  is the function that maps a binary encoded integer matrix to the binary encoding of its determinant, see [10]. Actually, Cook originally defined DET as  $\text{NC}^1(\text{det})$  [10], but later [11] remarked that the above definition via  $\text{AC}^0$ -circuits seems to be more natural. For instance, it implies that DET is equal to the #L-hierarchy.

We defined DET as a function class, but the definition can be extended to languages by considering their characteristic functions. It is well known that  $\text{NL} \subseteq \text{DET} \subseteq \text{NC}^2$  [11]. From  $\text{NL} \subseteq \text{DET}$ , it follows easily that  $\text{AC}^0(\text{NL}, f_1, \dots, f_k) \subseteq \text{DET}$  whenever  $f_1, \dots, f_k \in \text{DET}$ .

### 3 Algebraic structures, semigroups, and semirings

An *algebraic structure*  $\mathcal{A} = (D, f_1, \dots, f_k)$  consists of a non-empty *domain*  $D$  and operations  $f_i : D^{n_i} \rightarrow D$  for  $1 \leq i \leq k$ . We often identify the domain with the structure, if it is clear from the context. A *substructure* of  $\mathcal{A}$  is a subset  $B \subseteq D$  that is closed under each of the operations  $f_i$ . We identify  $B$  with the structure  $(B, g_1, \dots, g_k)$ , where  $g_i : B^{n_i} \rightarrow B$  is the restriction of  $f_i$  to  $B^{n_i}$  for all  $1 \leq i \leq k$ . We mainly deal with semigroups and semirings. In the following two subsection we present the necessary background. For further details concerning semigroup theory (resp., semiring theory) see [28] (resp., [13]).

#### 3.1 Semigroups

A *semigroup*  $(S, \circ)$  (or briefly  $S$ ) is an algebraic structure with a single associative binary operation. We usually write  $st$  for  $s \circ t$ . If  $st = ts$  for all  $s, t \in S$ , we call  $S$  *commutative*. A set  $I \subseteq S$  is called a *semigroup ideal* if for all  $s \in S, a \in I$  we have  $sa, as \in I$ . An element  $e \in S$  is called *idempotent* if  $ee = e$ . It is well-known that for every finite semigroup  $S$  and  $s \in S$  there exists an  $n \geq 1$  such that  $s^n$  is idempotent. In particular, every finite semigroup contains an idempotent element. By taking the smallest common multiple of all these  $n$ , one obtains an  $\omega \geq 1$  such that  $s^\omega$  is idempotent for all  $s \in S$ . The set of all idempotents of  $S$  is denoted with  $E(S)$ . If  $S$  is finite, then  $SE(S)S = S^n$  where  $n = |S|$ . Moreover,  $S^n = S^m$  for all  $m \geq n$ .

A semigroup  $M$  with an identity element  $1 \in M$ , i.e.  $1m = m1 = m$  for all  $m \in M$ , is called a *monoid*. With  $S^1$  we denote the monoid that is obtained from a semigroup  $S$  by adding a fresh element 1, which becomes the identity element of  $S^1$  by setting  $1s = s1 = s$  for all  $s \in S \cup \{1\}$ . In case  $M$  is a monoid and  $N$  is a submonoid of  $M$ , we do not require that the identity element of  $N$  is the identity element of  $M$ . But, clearly, the identity element of the submonoid  $N$  must be an idempotent element of  $M$ . In fact, for every semigroup  $S$  and every idempotent  $e \in E(S)$ , the set  $eSe = \{ese \mid s \in S\}$  is a submonoid of  $S$  with identity  $e$ , which is also called a *local submonoid* of  $S$ . The local submonoid  $eSe$  is the maximal submonoid of  $S$  whose identity element is  $e$ . A semigroup  $S$  is *aperiodic* if every subgroup of  $S$  is trivial. A semigroup  $S$  is *solvable* if every subgroup  $G$  of  $S$  is a solvable group, i.e., repeatedly taking the commutator subgroup leads from  $G$  to 1. Since Abelian groups are solvable, every commutative semigroup is solvable.

### 3.2 Semirings

A *semiring*  $(R, +, \cdot)$  consists of a non-empty set  $R$  with two operations  $+$  and  $\cdot$  such that  $(R, +)$  is a commutative semigroup,  $(R, \cdot)$  is a semigroup, and  $\cdot$  left- and right-distributes over  $+$ , i.e.,  $a \cdot (b + c) = ab + ac$  and  $(b + c) \cdot a = ba + ca$  (as usual, we write  $ab$  for  $a \cdot b$ ). Note that we neither require the existence of an additive identity  $0$  nor the existence of a multiplicative identity  $1$ . We denote with  $R_+ = (R, +)$  the additive semigroup of  $R$  and with  $R_\bullet = (R, \cdot)$  the multiplicative semigroup of  $R$ . For  $n \geq 1$  and  $r \in R$  we write  $n \cdot r$  or just  $nr$  for  $r + \dots + r$ , where  $r$  is added  $n$  times. For a non-empty subset  $T \subseteq R$  we denote by  $\langle T \rangle$  the subsemiring generated by  $T$ , i.e., the smallest set containing  $T$  which is closed under addition and multiplication. An *ideal* of  $R$  is a subset  $I \subseteq R$  such that for all  $a, b \in I, s \in R$  we have  $a + b, sa, as \in I$ . Clearly, every ideal is a subsemiring. With  $E(R)$  we denote the set of multiplicative idempotents of  $R$ , i.e., those  $e \in R$  with  $e^2 = e$ . Note that for every multiplicative idempotent  $e \in E(R)$ ,  $eRe$  is a subsemiring of  $R$  in which the multiplicative structure is a monoid. Let  $\mathbb{B}_2 = (\{0, 1\}, \vee, \wedge)$  be the *Boolean semiring*.

A crucial definition in this paper is that of a  $\{0, 1\}$ -free semiring. This is a semiring  $R$  which does *not* contain a subsemiring  $T$  with an additive identity  $0$  and a multiplicative identity  $1 \neq 0$ . Note that it is not required that  $0$  is absorbing in  $T$ , i.e.,  $a \cdot 0 = 0 \cdot a = 0$  for all  $a \in T$ . The class of  $\{0, 1\}$ -free *finite* semirings has several characterizations:

► **Lemma 1.** *For a finite semiring  $R$ , the following are equivalent:*

1.  $R$  is not  $\{0, 1\}$ -free.
2.  $\mathbb{B}_2$  or  $\mathbb{Z}_d$  for some  $d \geq 2$  is a subsemiring of  $R$ .
3.  $\mathbb{B}_2$  or  $\mathbb{Z}_d$  for some  $d \geq 2$  is a homomorphic image of a subsemiring of  $R$ .
4. There exist elements  $0, 1 \in R$  such that  $0 \neq 1$ ,  $0 + 0 = 0$ ,  $0 + 1 = 1$ ,  $0 \cdot 1 = 1 \cdot 0 = 0 \cdot 0 = 0$ , and  $1 \cdot 1 = 1$  (but  $1 + 1 \neq 1$  is possible).

As a consequence of Lemma 1 (point 4), one can check in time  $O(n^2)$  for a semiring of size  $n$  whether it is  $\{0, 1\}$ -free. We will not need this fact, since in our setting the semiring will be always fixed, i.e., not part of the input. Moreover, the class of all  $\{0, 1\}$ -free semirings is a pseudo-variety of finite semirings, i.e., it is closed under taking subsemirings (this is trivial), taking homomorphic images (by point 3), and direct products. For the latter, assume that  $R \times R'$  is not  $\{0, 1\}$ -free. Hence, there exists a subsemiring  $T$  of  $R \times R'$  with an additive zero  $(0, 0')$  and a multiplicative one  $(1, 1') \neq (0, 0')$ . W.l.o.g. assume that  $0 \neq 1$ . Then the projection  $\pi_1(T)$  onto the first component is a subsemiring of  $R$ , where  $0$  is an additive identity and  $1 \neq 0$  is a multiplicative identity.

## 4 Circuit evaluation and main results

We define circuits over general algebraic structures. Let  $\mathcal{A} = (D, f_1, \dots, f_k)$  be an algebraic structure. A *circuit* over  $\mathcal{A}$  is a triple  $\mathcal{C} = (V, A_0, \text{rhs})$  where  $V$  is a finite set of *gates*,  $A_0 \in V$  is the *output gate* and *rhs* (for right-hand side) is a function that assigns to each gate  $A \in V$  an element  $a \in D$  or an expression of the form  $f_i(A_1, \dots, A_n)$ , where  $n = n_i$  and  $A_1, \dots, A_n \in V$  are called the *input gates for  $A$* . Moreover, the binary relation  $\{(A, B) \in V \times V \mid A \text{ is an input gate for } B\}$  must be acyclic. The reflexive and transitive closure of it is a partial order on  $V$  that we denote with  $\leq_{\mathcal{C}}$ . Every gate  $A$  evaluates to an element  $[A]_{\mathcal{C}} \in \mathcal{A}$  in the natural way: If  $\text{rhs}(A) = a \in D$ , then  $[A]_{\mathcal{C}} = a$  and if  $\text{rhs}(A) = f_i(A_1, \dots, A_n)$  then  $[A]_{\mathcal{C}} = f_i([A_1]_{\mathcal{C}}, \dots, [A_n]_{\mathcal{C}})$ . Moreover, we define  $[\mathcal{C}] = [A_0]_{\mathcal{C}}$  (the value computed by  $\mathcal{C}$ ). If the circuit  $\mathcal{C}$  is clear from the context, we also write  $[A]$  instead of  $[A]_{\mathcal{C}}$ . Two circuits  $\mathcal{C}_1$  and  $\mathcal{C}_2$  over the structure  $\mathcal{A}$  are *equivalent* if  $[\mathcal{C}_1] = [\mathcal{C}_2]$ .

Sometimes we also use circuits without an output gate; such a circuit is just a pair  $(V, \text{rhs})$ . A subcircuit of  $\mathcal{C}$  is the restriction of  $\mathcal{C}$  to a downwards closed (w.r.t.  $\leq_{\mathcal{C}}$ ) subset of  $V$ . A gate  $A$  with  $\text{rhs}(A) = f_i(A_1, \dots, A_n)$  is called an *inner gate*, otherwise it is an *input gate* of  $\mathcal{C}$ . Quite often, we view a circuit as a directed acyclic graph, where the inner nodes are labelled with an operations  $f_i$ , and the leaf nodes are labelled with elements from  $D$ . In our proofs, it is sometimes convenient to allow arbitrary terms built from  $V \cup D$  using the operations  $f_1, \dots, f_k$  in right-hand sides. For instance, over a semiring  $(R, +, \cdot)$  we might have  $\text{rhs}(A) = s \cdot B \cdot t + C + s$  for  $s, t \in R$  and  $B, C \in V$ . A circuit is in *normal form*, if all right-hand sides are of the form  $a \in D$  or  $f_i(A_1, \dots, A_n)$  with  $A_1, \dots, A_n \in V$ . We will make use of the following simple fact:

► **Lemma 2.** *A circuit can be transformed in logspace into an equivalent normal form circuit.*

The *circuit evaluation problem*  $\text{CEP}(\mathcal{A})$  for some algebraic structure  $\mathcal{A}$  (say a semigroup or a semiring) is the following computational problem:

**Input:** A circuit  $\mathcal{C}$  over  $\mathcal{A}$  and an element  $a \in D$  from its domain.

**Output:** Decide whether  $[\mathcal{C}] = a$ .

Note that for a finite structure  $\mathcal{A}$ ,  $\text{CEP}(\mathcal{A})$  is basically equivalent to its computation variant, where one actually computes the output value  $[\mathcal{C}]$  of the circuit: if  $\text{CEP}(\mathcal{A})$  belongs to a complexity class  $\mathsf{C}$ , then the computation variant belongs to  $\text{AC}^0(\mathsf{C})$ , and if the latter belongs to  $\text{AC}^0(\mathsf{C})$  then  $\text{CEP}(\mathcal{A})$  belongs to the decision fragment of  $\text{AC}^0(\mathsf{C})$ .

Clearly, for every finite structure the circuit evaluation problem can be solved in polynomial time by evaluating all gates along the partial order  $\leq_{\mathcal{C}}$ . Ladner's classical P-completeness result for the Boolean circuit value problem [19] can be stated as follows:

► **Theorem 3** ([19]).  *$\text{CEP}(\mathbb{B}_2)$  is P-complete.*

For semigroups, the following dichotomy was shown in [8]:

- **Theorem 4** ([8]). *Let  $S$  be a finite semigroup.*
- *If  $S$  is aperiodic, then  $\text{CEP}(S)$  is in NL.*
- *If  $S$  is solvable, then  $\text{CEP}(S)$  belongs to DET.*
- *If  $S$  is not solvable, then  $\text{CEP}(S)$  is P-complete.*

In fact, in [8], the authors use the original definition  $\text{DET} = \text{NC}^1(\text{det})$  of Cook. But the arguments in [8] actually show that for a finite solvable semigroup,  $\text{CEP}(S)$  belongs to  $\text{AC}^0(\text{det})$  (which is our definition of DET). Moreover, in [8], Theorem 4 is only shown for monoids, but the extension to semigroups is straightforward: If the finite semigroup  $S$  has a non-solvable subgroup, then  $\text{CEP}(S)$  is P-complete, since the circuit evaluation problem for a non-solvable finite group is P-complete. On the other hand, if  $S$  is solvable (resp., aperiodic), then also the monoid  $S^1$  is solvable (resp., aperiodic). This holds, since the subgroups of  $S^1$  are exactly the subgroups of  $S$  together with  $\{1\}$ . Hence,  $\text{CEP}(S^1)$  is in DET (resp., NL), which implies that  $\text{CEP}(S)$  is in DET (resp., NL).

Let us fix a *finite* semiring  $R = (R, +, \cdot)$  for the rest of the paper. Note that  $\text{CEP}(R_+)$  (resp.,  $\text{CEP}(R_\bullet)$ ) is the restriction of  $\text{CEP}(R)$  to circuits without multiplication (resp., addition) gates. Since every commutative semigroup is solvable, Theorem 4 implies that  $\text{CEP}(R_+)$  belongs to DET. The main result of this paper is:

► **Theorem 5.** *If the finite semiring  $R$  is  $\{0, 1\}$ -free, then the problem  $\text{CEP}(R)$  belongs to the class  $\text{AC}^0(\text{NL}, \text{CEP}(R_+), \text{CEP}(R_\bullet))$ . Otherwise  $\text{CEP}(R)$  is P-complete.*

Note that  $\text{CEP}(R)$  can also be P-complete for a  $\{0, 1\}$ -free semiring (namely in the case that  $\text{CEP}(R_\bullet)$  is P-complete) and that  $\text{AC}^0(\text{NL}, \text{CEP}(R_+), \text{CEP}(R_\bullet)) = \text{AC}^0(\text{CEP}(R_+), \text{CEP}(R_\bullet))$  whenever  $\text{CEP}(R_+)$  or  $\text{CEP}(R_\bullet)$  is NL-hard. For example, this is the case, if  $R_+$  or  $R_\bullet$  is an aperiodic nontrivial monoid [8, Proposition 4.14] (for aperiodic nontrivial monoids one can easily reduce the NL-complete of graph reachability problem to the circuit value problem).

The P-hardness statement in Theorem 5 is easy to show:

► **Lemma 6.** *If the finite semiring  $R$  is not  $\{0, 1\}$ -free, then  $\text{CEP}(R)$  is P-complete.*

**Proof.** By Lemma 1,  $R$  contains either  $\mathbb{B}_2$  or  $\mathbb{Z}_d$  for some  $d \geq 2$ . In the former case, P-hardness follows from Ladner's theorem. Furthermore, one can reduce the P-complete Boolean circuit value problem over  $\{0, 1, \wedge, \neg\}$  to  $\text{CEP}(\mathbb{Z}_d)$ : A gate  $z = x \wedge y$  is replaced by  $z = x \cdot y$  and a gate  $y = \neg x$  is replaced by  $y = 1 + (d - 1) \cdot x$ . ◀

Theorem 4 and 5 yield the following corollaries:

► **Corollary 7.** *Let  $R$  be a finite semiring.*

- *If  $R$  is  $\{0, 1\}$ -free and  $R_\bullet$  and  $R_+$  are aperiodic, then  $\text{CEP}(R)$  belongs to NL.*
- *If  $R$  is  $\{0, 1\}$ -free and  $R_\bullet$  is solvable, then  $\text{CEP}(R)$  belongs to DET.*
- *If  $R$  is not  $\{0, 1\}$ -free or  $R_\bullet$  is not solvable, then  $\text{CEP}(R)$  is P-complete.*

Let us present an application of Corollary 7.

► **Example 8.** An important semigroup construction found in the literature is the power construction. For a finite semigroup  $S$  one defines the *power semiring*  $\mathcal{P}(S) = (2^S \setminus \{\emptyset\}, \cup, \cdot)$  with the multiplication  $A \cdot B = \{ab \mid a \in A, b \in B\}$ . Notice that if one includes the empty set, then the semiring would not be  $\{0, 1\}$ -free: Take an idempotent  $e \in S$ . Then  $\emptyset$  and  $\{e\}$  form a copy of  $\mathbb{B}_2$ . Hence, the circuit evaluation problem is P-complete.

Let us further assume that  $S$  is a monoid with identity 1 (the general case will be considered below). If  $S$  contains an idempotent  $e \neq 1$  then also  $\mathcal{P}(S)$  is not  $\{0, 1\}$ -free:  $\{e\}$  and  $\{1, e\}$  form a copy of  $\mathbb{B}_2$ . On the other hand, if 1 is the unique idempotent of  $S$ , then  $S$  must be a group  $G$ . Assume that  $G$  is solvable; otherwise  $\mathcal{P}(G)_\bullet$  is not solvable as well and has a P-complete circuit evaluation problem by Theorem 4. It is not hard to show that the subgroups of  $\mathcal{P}(G)_\bullet$  correspond to the quotient groups of subgroups of  $G$ ; see also [21]. Since  $G$  is solvable and the class of solvable groups is closed under taking subgroups and quotients,  $\mathcal{P}(G)_\bullet$  is a solvable monoid. Moreover  $\mathcal{P}(G)$  is  $\{0, 1\}$ -free: Otherwise, Lemma 1 implies that there are non-empty subsets  $A, B \subseteq G$  such that  $A \neq B$ ,  $A \cup B = B$  (and thus  $A \subsetneq B$ ),  $AB = BA = A^2 = A$ , and  $B^2 = B$ . Hence,  $B$  is a subgroup of  $G$  and  $A \subseteq B$ . But then  $B = AB = A$ , which is a contradiction. By Corollary 7,  $\text{CEP}(\mathcal{P}(G))$  for a finite solvable group  $G$  belongs to DET.

Let us now classify the complexity of  $\text{CEP}(\mathcal{P}(S))$  for arbitrary semigroups  $S$ . A semigroup  $S$  is a *local group* if for all  $e \in E(S)$  the local monoid  $eSe$  is a group. In a finite local group  $S$  of size  $n$  the minimal semigroup ideal is  $S^n = SE(S)S$  [2, Proposition 2.3].

► **Theorem 9.** *Let  $S$  be a finite semigroup. If  $S$  is a local group and solvable, then  $\text{CEP}(\mathcal{P}(S))$  belongs to DET. Otherwise  $\text{CEP}(\mathcal{P}(S))$  is P-complete.*

**Proof.** If  $S$  is a solvable local group, then the multiplicative semigroup  $\mathcal{P}(S)_\bullet$  is solvable as well [5, Corollary 2.7]. It remains to show that the semiring  $\mathcal{P}(S)$  is  $\{0, 1\}$ -free. Towards a contradiction assume that  $\mathcal{P}(S)$  is not  $\{0, 1\}$ -free. By Lemma 1, there exist non-empty sets  $A \subsetneq B \subseteq S$  such that  $AB = BA = A^2 = A$  and  $B^2 = B$ . Hence,  $B$  is a subsemigroup of  $S$ ,

which is also a local group, and  $A$  is a semigroup ideal in  $B$ . Since the minimal semigroup ideal of  $B$  is  $B^n$  for  $n = |B|$  and  $B^n = B$ , we obtain  $A = B$ , which is a contradiction.

If  $S$  is not a local group, then there exists a local monoid  $eSe$  which is not a group and hence contains an idempotent  $f \neq e$ . Since  $\{\{f\}, \{e, f\}\}$  forms a copy of  $\mathbb{B}_2$  it follows that  $\text{CEP}(\mathcal{P}(S))$  is P-complete. Finally, if  $S$  is not solvable, then also  $\mathcal{P}(S)$  is not solvable and  $\text{CEP}(\mathcal{P}(S))$  is P-complete by Theorem 4.  $\blacktriangleleft$

## 5 Proof of Theorem 5

The proof of Theorem 5 will proceed in two steps. In the first step we reduce the problem to evaluating circuits in which the computation admits a type-function defined in the following. In the second step, we show how to evaluate such circuits.

► **Definition 10.** Let  $E = E(R)$  be the set of multiplicative idempotents. Let  $\mathcal{C} = (V, \text{rhs})$  be a circuit in normal form such that  $[A]_{\mathcal{C}} \in ERE$  for all  $A \in V$ . A type-function for  $\mathcal{C}$  is a mapping  $\text{type} : V \rightarrow E \times E$  such that for all gates  $A \in V$ :

- If  $\text{type}(A) = (e, f)$ , then  $[A]_{\mathcal{C}} \in eRf$ .
- If  $A$  is an addition gate with  $\text{rhs}(A) = B + C$ , then  $\text{type}(A) = \text{type}(B) = \text{type}(C)$ .
- If  $A$  is a multiplication gate with  $\text{rhs}(A) = B \cdot C$ ,  $\text{type}(B) = (e, e')$ , and  $\text{type}(C) = (f', f)$ , then  $\text{type}(A) = (e, f)$ .

A circuit is called *type admitting* if it admits a type-function.

A function  $\alpha : R^m \rightarrow R$  ( $m \geq 0$ ) is called *affine* if there are  $a_1, b_1, \dots, a_m, b_m, c \in R$  such that  $\alpha(x_1, \dots, x_m) = \sum_{i=1}^m a_i x_i b_i + c$  or  $\alpha(x_1, \dots, x_m) = \sum_{i=1}^m a_i x_i b_i$  for all  $x_1, \dots, x_m \in R$ . We represent this affine function by the tuple  $(a_1, b_1, \dots, a_m, b_m, c)$  or  $(a_1, b_1, \dots, a_m, b_m)$ . Theorem 5 is an immediate corollary of the following two propositions (and the obvious fact that an affine function with a constant number of inputs can be evaluated in  $\text{AC}^0$ ).

► **Proposition 11.** *Given a circuit  $\mathcal{C}$  over the finite semiring  $R$ , one can compute in  $\text{AC}^0(\text{NL}, \text{CEP}(R_+))$*

- an affine function  $\alpha : R^m \rightarrow R$  for some  $0 \leq m \leq |R|^4$ ,
- a type admitting circuit  $\mathcal{C}' = (V', \text{rhs}')$ , and
- a list of gates  $A_1, \dots, A_m \in V'$  such that  $[\mathcal{C}] = \alpha([A_1]_{\mathcal{C}'}, \dots, [A_m]_{\mathcal{C}'})$ .

► **Proposition 12.** *If  $R$  is  $\{0, 1\}$ -free, then the restriction of  $\text{CEP}(R)$  to type admitting circuits is in  $\text{AC}^0(\text{NL}, \text{CEP}(R_+), \text{CEP}(R_\bullet))$ .*

Notice that in Proposition 12 we do not need explicitly a type function as part of the input. Moreover, it is not clear how to test efficiently whether a circuit is type admitting. On the other hand, this is not a problem for us, since we will apply Proposition 12 only to circuits resulting from Proposition 11, which are type admitting by construction.

### 5.1 Step 1: Reduction to typing admitting circuits

In this section, we sketch a proof of Proposition 11. Let  $\mathcal{C}$  be a circuit in normal form over our fixed finite semiring  $(R, +, \cdot)$  of size  $n = |R| \geq 2$  (the case  $n = 1$  is trivial). Let  $E = E(R)$ . Note that  $R^n = RER$  is closed under multiplication with elements from  $R$ . Thus,  $\langle R^n \rangle$  is an ideal. Every element of  $\langle R^n \rangle$  is a finite sum of elements from  $R^n$ .

In a first step, we compute from  $\mathcal{C}$  in  $\text{AC}^0(\text{NL}, \text{CEP}(R_+))$  a semiring element  $r$  and a circuit  $\mathcal{D}$  over the subsemiring  $\langle R^n \rangle = \langle RER \rangle$  such that  $[\mathcal{C}] = r + [\mathcal{D}]$ , where  $r$  or  $\mathcal{D}$  (but not both) can be missing. For the proof of this, we interpret the circuit  $\mathcal{C}$  over the *free*



semiring  $\mathbb{N}[R]$ . It consists of all mappings  $f : R^+ \rightarrow \mathbb{N}$  (where  $R^+$  is the set of non-empty words over the alphabet  $R$ ) such that  $\text{supp}(f) := \{w \in R^+ \mid f(w) \neq 0\}$  (the support of  $f$ ) is finite and non-empty. We view an element  $f \in \mathbb{N}[R]$  as a polynomial  $\sum_{w \in \text{supp}(f)} f(w) \cdot w$ , where  $R$  is a set of non-commuting variables. Addition and multiplication of such non-commuting polynomials is defined as usual. Words  $w \in \text{supp}(f)$  are also called *monomials* of  $f$ . Let  $h : \mathbb{N}[R] \rightarrow R$  be the canonical evaluation homomorphism, which evaluates a given non-commutative polynomial in  $R$ . Thereby a monomial  $w = a_1 a_2 \cdots a_n$  is mapped to the corresponding product in  $R$ . Since a semiring is not assumed to have a multiplicative identity (resp., additive identity), we have to exclude the empty word from  $\text{supp}(f)$  for every  $f \in \mathbb{N}[R]$  (resp., exclude the mapping  $f$  with  $\text{supp}(f) = \emptyset$  from  $\mathbb{N}[R]$ ).

The idea is to split each polynomial computed in a gate  $A$  into two parts: Those monomials (i.e., non-empty words over  $R$ ) that have length  $< n = |R|$  (called the short part of  $A$ ) and those monomials that have length  $\geq n$  (called the long part of  $A$ ). Of course the short (resp. long) part of a gate can be empty. We then compute from the circuit  $\mathcal{C}$  the following data: (i) for every gate  $A$  the  $h$ -image of the short part of  $A$  if it is non-empty and (ii) a circuit over  $\langle R^n \rangle$  that contains for every gate  $A$  of  $\mathcal{C}$  the  $h$ -image of its long part (if it exists). For (i), we need oracle access to  $\text{CEP}(R_+)$ . Oracle access to  $\text{NL}$  is needed to compute those gates whose short (resp., long) part is non-empty.

In a second step, we compute from a circuit  $\mathcal{D}$  over  $\langle RER \rangle$  a type admitting circuit  $\mathcal{C}'$  such that the value of  $\mathcal{D}$  is an affine combination of certain gate values in  $\mathcal{C}'$ . The main idea is the following: In the circuit  $\mathcal{D}$  all input values are sums of elements of the form  $set$  ( $e \in E$ ,  $s, t \in R$ ), which we can write as  $se^3t$ . Hence, if we evaluate the circuit freely in  $\mathbb{N}[R]$ , then every monomial that arises at a gate  $A$  is of the form  $segft$ , where  $g$  starts (resp., ends) with the symbol  $e \in E$  (resp.,  $f \in E$ ) and  $s, t \in R$ . Let  $P_A$  is the set of all tuples  $(s, e, f, t)$  such that at gate  $A$  a monomial of the form  $segft$  arises. One can show that  $P_A$  can be computed in  $\text{AC}^0(\text{NL})$ . The circuit  $\mathcal{C}'$  contains for every  $(s, e, f, t) \in P_A$  a gate  $A_{s,e,f,t}$  that computes the sum of all monomials  $g$  such that  $segft$  is a monomial that appears at gate  $A$ . The type of gate  $A_{s,e,f,t}$  is  $(e, f)$ . Moreover,  $[A]_{\mathcal{D}}$  is equal to  $\sum_{(s,e,f,t) \in P_A} (se)[A_{s,e,f,t}]_{\mathcal{C}'}(ft)$ . This shows that  $[D]$  is indeed an affine combination of certain gate values in  $\mathcal{C}'$ .

## 5.2 Step 2: A parallel evaluation algorithm for type admitting circuits

In this section we prove Proposition 12. We present a parallel evaluation algorithm for type admitting circuits. This algorithm terminates after at most  $|R|$  rounds, if  $R$  has a so-called rank-function, which we define first. As before, let  $E = E(R)$ .

► **Definition 13.** We call a function  $\text{rank} : R \rightarrow \mathbb{N} \setminus \{0\}$  a *rank-function* for  $R$  if it satisfies the following conditions for all  $a, b \in R$ :

1.  $\text{rank}(a) \leq \text{rank}(a \circ b)$  and  $\text{rank}(b) \leq \text{rank}(a \circ b)$  for  $\circ \in \{+, \cdot\}$ .
2. If  $a, b \in eRf$  for some  $e, f \in E$  and  $\text{rank}(a) = \text{rank}(a + b)$ , then  $a = a + b$ .

If  $R_\bullet$  is a monoid, then one can choose  $e = 1 = f$  in the second condition in Definition 13, which is therefore equivalent to: If  $\text{rank}(a) = \text{rank}(a + b)$  for  $a, b \in R$ , then  $a = a + b$ .

► **Example 14 (Example 8 continued).** Let  $G$  be a finite group and consider the semiring  $\mathcal{P}(G)$ . One can verify that the function  $A \mapsto |A|$ , where  $\emptyset \neq A \subseteq G$ , is a rank-function for  $\mathcal{P}(G)$ . On the other hand, if  $S$  is a finite semigroup, which is not a group, then  $S$  cannot be cancellative. Assume that  $ab = ac$  for  $a, b, c \in S$  with  $b \neq c$ . Then  $\{a\} \cdot \{b, c\} = \{ab\}$ . This shows that the function  $A \mapsto |A|$  is not a rank-function for  $\mathcal{P}(S)$ .

► **Theorem 15.** *If the finite semiring  $R$  has a rank-function  $\text{rank}$ , then the restriction of  $\text{CEP}(R)$  to type admitting circuits belongs to  $\text{AC}^0(\text{NL}, \text{CEP}(R_+), \text{CEP}(R_\bullet))$ .*

**Proof.** Let  $\mathcal{C} = (V, A_0, \text{rhs})$  be a circuit with the type function  $\text{type}$ . We present an algorithm which partially evaluates the circuit in a constant number of phases, where each phase can be carried out in  $\text{AC}^0(\text{NL}, \text{CEP}(R_+), \text{CEP}(R_\bullet))$  and the following invariant is preserved:

**Invariant:** After phase  $k$  all gates  $A$  with  $\text{rank}([A]_{\mathcal{C}}) \leq k$  are evaluated, i.e., are input gates in phase  $k + 1$  onwards.

Initially, i.e., for  $k = 0$ , the invariant holds, since 0 is not in the range of the rank-function. After  $\max\{\text{rank}(a) \mid a \in R\}$  (which is a constant) many phases, the output gate  $A_0$  is evaluated. We present phase  $k$  of the algorithm, assuming that the invariant holds after phase  $k - 1$ . Thus, all gates  $A$  with  $\text{rank}([A]_{\mathcal{C}}) < k$  of the current circuit  $\mathcal{C}$  are input gates. In phase  $k$  we evaluate all gates  $A$  with  $\text{rank}([A]_{\mathcal{C}}) = k$ . For this, we proceed in two steps:

**Step 1.** As a first step the algorithm evaluates all subcircuits that only contain addition and input gates. This maintains the invariant and is possible in  $\text{AC}^0(\text{NL}, \text{CEP}(R_+))$ . After this step, every addition-gate  $A$  has at least one inner input gate, which we denote by  $\text{inner}(A)$  (if both input gates are inner gates, then choose one arbitrarily). The NL-oracle access is needed to compute the set of all gates  $A$  for which no multiplication gate  $B \leq_{\mathcal{C}} A$  exists.

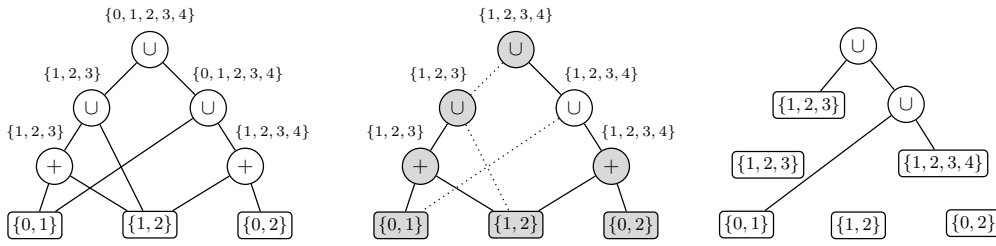
**Step 2.** Define the multiplicative circuit  $\mathcal{C}' = (V, A_0, \text{rhs}')$  by

$$\text{rhs}'(A) = \begin{cases} \text{inner}(A) & \text{if } A \text{ is an addition-gate,} \\ \text{rhs}(A) & \text{if } A \text{ is a multiplication gate or input gate.} \end{cases} \quad (1)$$

The circuit  $\mathcal{C}'$  can be brought in logspace into normal form by Lemma 2 and then evaluated in  $\text{AC}^0(\text{CEP}(R_\bullet))$ . A gate  $A \in V$  is called *locally correct* if (i)  $A$  is an input gate or multiplication gate of  $\mathcal{C}$ , or (ii)  $A$  is an addition gate of  $\mathcal{C}$  with  $\text{rhs}(A) = B + C$  and  $[A]_{\mathcal{C}'} = [B]_{\mathcal{C}'} + [C]_{\mathcal{C}'}$ . We compute the set  $W := \{A \in V \mid \text{all gates } B \text{ with } B \leq_{\mathcal{C}} A \text{ are locally correct}\}$  in  $\text{AC}^0(\text{NL})$ . A simple induction shows that for all  $A \in W$  we have  $[A]_{\mathcal{C}} = [A]_{\mathcal{C}'}$ . Hence we can set  $\text{rhs}(A) = [A]_{\mathcal{C}'}$  for all  $A \in W$ . This concludes phase  $k$  of the algorithm.

To prove that the invariant holds after phase  $k$ , we show that for each gate  $A \in V$  with  $\text{rank}([A]_{\mathcal{C}}) \leq k$  we have  $A \in W$ . This is shown by induction over the depth of  $A$  in  $\mathcal{C}$ . Assume that  $\text{rank}([A]_{\mathcal{C}}) \leq k$ . By the first condition from Definition 13, all gates  $B <_{\mathcal{C}} A$  satisfy  $\text{rank}([B]_{\mathcal{C}}) \leq k$ . Thus, the induction hypothesis yields  $B \in W$  and hence  $[B]_{\mathcal{C}} = [B]_{\mathcal{C}'}$  for all gates  $B <_{\mathcal{C}} A$ . It remains to show that  $A$  is locally correct, which is clear if  $A$  is an input gate or a multiplication gate. So assume that  $\text{rhs}(A) = B + C$  where  $B = \text{inner}(A)$ , which implies  $[A]_{\mathcal{C}'} = [B]_{\mathcal{C}'}$  by (1). Since  $B$  is an inner gate, which is not evaluated after phase  $k - 1$ , it holds that  $\text{rank}([B]_{\mathcal{C}}) \geq k$  and therefore  $\text{rank}([A]_{\mathcal{C}}) = \text{rank}([B]_{\mathcal{C}}) = k$ . By Definition 10 there exist idempotents  $e, f \in E$  with  $\text{type}(B) = \text{type}(C) = (e, f)$  and thus  $[B]_{\mathcal{C}}, [C]_{\mathcal{C}} \in eRf$ . The second condition from Definition 13 implies that  $[A]_{\mathcal{C}} = [B]_{\mathcal{C}} + [C]_{\mathcal{C}} = [B]_{\mathcal{C}}$ . We finally get  $[A]_{\mathcal{C}'} = [B]_{\mathcal{C}'} = [B]_{\mathcal{C}} = [A]_{\mathcal{C}} = [B]_{\mathcal{C}} + [C]_{\mathcal{C}} = [B]_{\mathcal{C}'} + [C]_{\mathcal{C}'}$ . Therefore  $A$  is locally correct. ◀

► **Example 16** (Example 8 continued). Figure 1 shows a circuit  $\mathcal{C}$  over the power semiring  $\mathcal{P}(G)$  of the group  $G = (\mathbb{Z}_5, +)$ . Recall from Example 14 that the function  $A \mapsto |A|$  is a rank function for  $\mathcal{P}(G)$ . We illustrate one phase of the algorithm. All gates  $A$  with  $\text{rank}([A]) < 3$  are evaluated in the circuit  $\mathcal{C}$  shown on the left. The goal is to evaluate all gates  $A$  with  $\text{rank}([A]) = 3$ . The first step would be to evaluate maximal  $\cup$ -circuits, which is already done.



■ **Figure 1** The parallel evaluation algorithm over the power semiring  $\mathcal{P}(\mathbb{Z}_5)$ .

In the second step the circuit  $\mathcal{C}'$  (shown in the middle) from the proof of Theorem 15 is computed and evaluated using the oracle for  $\text{CEP}(\mathbb{Z}_5, +)$ . The dotted wires do not belong to the circuit  $\mathcal{C}'$ . All locally correct gates are shaded. Note that the output gate is locally correct but its right child is not locally correct. All other shaded gates form a downwards closed set, which is the set  $W$  from the proof. These gates can be evaluated such that in the resulting circuit (shown on the right) all gates which evaluate to elements of rank 3 are evaluated.

To show Proposition 12, it remains to equip every finite  $\{0, 1\}$ -free semiring with a rank-function.

► **Lemma 17.** *If  $R$  is  $\{0, 1\}$ -free and  $e, f \in E(R)$  are such that  $ef = fe = f + f = f$ , then  $e + f = f$ .*

**Proof.** With  $f = 0$ ,  $e + f = 1$  all equations from Lemma 1 (point 4) hold; hence  $e + f = f$ . ◀

► **Lemma 18.** *If the finite semiring  $R$  is  $\{0, 1\}$ -free, then  $R$  has a rank-function.*

**Proof.** For  $a, b \in R$  we define  $a \preceq b$  if  $b$  can be obtained from  $a$  by iterated additions and left- and right-multiplications of elements from  $R$ . This is equivalent to the existence of  $\ell, r, c \in R$  such that  $b = \ell ar + c$ , where each of the elements  $\ell, r, c$  can be missing. Since  $\preceq$  is a preorder on  $R$ , there is a function  $\text{rank} : R \rightarrow \mathbb{N} \setminus \{0\}$  such that for all  $a, b \in R$  we have (i)  $\text{rank}(a) = \text{rank}(b)$  if and only if  $a \preceq b \preceq a$ , and (ii)  $\text{rank}(a) \leq \text{rank}(b)$  if  $a \preceq b$ .

We claim that  $\text{rank}$  satisfies the conditions of Definition 13. The first condition is clear, since  $a \preceq a + b$  and  $a, b \preceq ab$ . For the second condition, let  $e, f \in E$ ,  $a, b \in eRf$  such that  $\text{rank}(a + b) = \text{rank}(a)$ , which is equivalent to  $a + b \preceq a$ . Assume that  $a = \ell(a + b)r + c = \ell ar + \ell br + c$  for some  $\ell, r, c \in R$  (the case without  $c$  can be handled in the same way). Since  $a = eaf$  and  $b = ebf$ , we have  $a = \ell e(a + b)fr + c$  and hence we can assume that  $\ell$  and  $r$  are not missing. Moreover,  $a = eaf = (ele)(a + b)(frf) + (ecf)$ , so we can assume that  $\ell = ele$  and  $r = frf$ . After  $m$  applications of  $a = \ell ar + \ell br + c$  we get

$$a = \ell^m ar^m + \sum_{i=1}^m \ell^i br^i + \sum_{i=0}^{m-1} \ell^i cr^i. \tag{2}$$

Let  $n \geq 1$  such that  $nx$  is additively idempotent and  $x^n$  is multiplicatively idempotent for all  $x \in R$ . Hence  $nx^n$  is both additively and multiplicatively idempotent for all  $x \in R$ . If we choose  $m = n^2$ , the right hand side of (2) contains the partial sum  $P := \sum_{i=1}^n \ell^{in} br^{in}$ . Furthermore,  $e(n\ell^n) = (n\ell^n)e = n\ell^n$  and  $f(nr^n) = (nr^n)f = nr^n$ . Therefore, Lemma 17

implies that  $n\ell^n = n\ell^n + e$  and  $nr^n = nr^n + f$ , and hence:

$$\begin{aligned} P &= \sum_{i=1}^n \ell^{in} b r^{in} = n(\ell^n b r^n) = n^2(\ell^n b r^n) = (n\ell^n) b (nr^n) = (n\ell^n + e) b (nr^n) \\ &= (n\ell^n) b (nr^n) + e b (nr^n) = (n\ell^n) b (nr^n) + e b (nr^n + f) \\ &= (n\ell^n) b (nr^n) + e b (nr^n) + e b f = \left( \sum_{i=1}^n \ell^{in} b r^{in} \right) + b = P + b. \end{aligned}$$

Thus, the partial sum  $P$  in (2) can be replaced by  $P + b$ , which shows  $a = a + b$ . ◀

## 6 An application to formal language theory

In this section we briefly report on an application of Corollary 7 to a particular intersection non-emptiness problem. We assume some familiarity with context-free grammars. A circuit over the free monoid  $\Sigma^*$  can be seen as a context-free grammar producing exactly one word. Such a circuit is also called a *straight-line program*, briefly SLP. It is an acyclic context-free grammar  $\mathcal{H}$  that contains for every non-terminal  $A$  exactly one rule with left-hand side  $A$ . We denote with  $\text{val}_{\mathcal{H}}(A)$  the unique terminal word that can be derived from  $A$ .

For an alphabet  $\Sigma$  and a language  $L \subseteq \Sigma^*$ , the *intersection non-emptiness problem for  $L$* , denoted by  $\text{CFG-IP}(L, \Sigma)$ , is the following decision problem: Given a context-free grammar  $\mathcal{G}$  over  $\Sigma$ , does  $L(\mathcal{G}) \cap L \neq \emptyset$  hold? For every regular language  $L$ , this problem belongs to P: One constructs in polynomial time a context-free grammar for  $L(\mathcal{G}) \cap L$  from  $\mathcal{G}$  and a finite automaton for  $L$  and tests this grammar for emptiness, which is possible in polynomial time. However, testing emptiness of a given context-free language is P-complete. An easy reduction shows that the problem  $\text{CFG-IP}(L, \Sigma)$  is P-complete for every  $L \neq \emptyset$ :

► **Theorem 19.** *For every non-empty language  $L \subseteq \Sigma^*$ ,  $\text{CFG-IP}(L, \Sigma)$  is P-complete.*

By Theorem 19 we have to put some restriction on context-free grammars in order to get NC-algorithms for intersection non-emptiness. It turns out that productivity of all non-terminals is the right assumption. Thus, we require that every non-terminal  $A$  is productive, i.e., a terminal word can be derived from  $A$ . In order to avoid a promise problem (testing productivity of a non-terminal is P-complete [16]) we add to the input grammar  $\mathcal{G}$  an SLP  $\mathcal{H}$ , which *uniformizes*  $\mathcal{G}$  in the sense that  $\mathcal{H}$  contains for every non-terminal  $A$  exactly one rule  $A \rightarrow \alpha$  from  $\mathcal{G}$ . Hence, the word  $\text{val}_{\mathcal{H}}(A)$  is a witness for the productivity of  $A$ . For instance, a uniformizing SLP for the grammar  $S \rightarrow SS \mid aSb \mid A$ ,  $A \rightarrow aA \mid B$ ,  $B \rightarrow bB \mid b$  would be  $S \rightarrow A$ ,  $A \rightarrow B$ ,  $B \rightarrow b$ .

We define the following restriction  $\text{PCFG-IP}(L, \Sigma)$  of  $\text{CFG-IP}(L, \Sigma)$ : Given a productive context-free grammar  $\mathcal{G}$  over  $\Sigma$  and a uniformizing SLP  $\mathcal{H}$  for  $\mathcal{G}$ , does  $L(\mathcal{G}) \cap L \neq \emptyset$  hold? The theorem below classifies regular languages  $L \subseteq \Sigma^*$  by the complexity of  $\text{PCFG-IP}(L, \Sigma)$ . To do this we use the standard notion of the syntactic monoid  $M_L$  of  $L$  (which is a finite monoid for  $L$  regular). There is a surjective morphism  $h : \Sigma^* \rightarrow M_L$  and a subset  $F \subseteq M_L$  such that  $L = h^{-1}(F)$ . Let us fix the regular language  $L \subseteq \Sigma^*$ ,  $M = M_L$ ,  $h : \Sigma^* \rightarrow M$  and  $F \subseteq M$ . Define the equivalence relation  $\sim_F$  on  $\mathcal{P}(M)$  by:  $A_1 \sim_F A_2$  ( $A_1, A_2 \in \mathcal{P}(M)$ ) if and only if  $\forall \ell, r \in M : \ell A_1 r \cap F \neq \emptyset \iff \ell A_2 r \cap F \neq \emptyset$ . It can be shown that  $\sim_F$  is a congruence relation. In particular,  $\mathcal{P}(M)/\sim_F$  is a semiring.

► **Theorem 20.**  *$\text{PCFG-IP}(L, \Sigma)$  is equivalent to  $\text{CEP}(\mathcal{P}(M)/\sim_F)$  with respect to constant depth reductions. Hence,  $\text{PCFG-IP}(L, \Sigma)$  is in DET (resp., NL) if  $(\mathcal{P}(M)/\sim_F)$  is solvable (resp., aperiodic) and  $\mathcal{P}(M)/\sim_F$  is  $\{0, 1\}$ -free; otherwise  $\text{PCFG-IP}(L, \Sigma)$  is P-complete.*

As an application of Theorem 20 one can show that  $\text{PCFG-IP}(L, \Sigma)$  is in NL for every language of the form  $L = \Sigma^* a_1 \Sigma^* a_2 \Sigma^* \dots a_k \Sigma^*$  for  $a_1, \dots, a_k \in \Sigma$ .

## 7 Conclusion and outlook

We proved a dichotomy result for the circuit evaluation problem for finite semirings: If (i) the semiring has no subsemiring with an additive and multiplicative identity and both are different and (ii) the multiplicative subsemigroup is solvable, then the circuit evaluation problem is in  $\text{DET} \subseteq \text{NC}^2$ , otherwise it is P-complete.

The ultimate goal would be to obtain such a dichotomy for all finite algebraic structures. One might ask whether for every finite algebraic structure  $\mathcal{A}$ ,  $\text{CEP}(\mathcal{A})$  is P-complete or in NC. It is known that under the assumption  $\text{P} \neq \text{NC}$  there exist problems in  $\text{P} \setminus \text{NC}$  that are not P-complete [32]. In [7] it is shown that every circuit evaluation problem  $\text{CEP}(\mathcal{A})$  is equivalent to a circuit evaluation problem  $\text{CEP}(A, \circ)$ , where  $\circ$  is a binary operation.

**Acknowledgement.** We are grateful to Ben Steinberg for fruitful discussions and to Volker Diekert for pointing out to us the proof of the implication  $(3 \Rightarrow 4)$  in the proof of Lemma 1.

---

### References

- 1 Eric Allender, Jia Jiao, Meena Mahajan, and V. Vinay. Non-commutative arithmetic circuits: Depth reduction and size lower bounds. *Theor. Comput. Sci.*, 209(1-2):47–86, 1998.
- 2 Jorge Almeida, Stuart Margolis, Benjamin Steinberg, and Mikhail Volkov. Representation theory of finite semigroups, semigroup radicals and formal language theory. *Transactions of the American Mathematical Society*, 361(3):1429–1461, 2009.
- 3 Carme Àlvarez, José L. Balcázar, and Birgit Jenner. Functional oracle queries as a measure of parallel time. In *Proceedings of the 8th Annual Symposium on Theoretical Aspects of Computer Science, STACS 1991*, volume 480 of *Lecture Notes in Computer Science*, pages 422–433. Springer, 1991.
- 4 Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach*. Cambridge University Press, 2009.
- 5 Karl Auinger and Benjamin Steinberg. Constructing divisions into power groups. *Theoretical Computer Science*, 341(1-3):1–21, 2005.
- 6 Martin Beaudry and Markus Holzer. The complexity of tensor circuit evaluation. *Computational Complexity*, 16(1):60–111, 2007.
- 7 Martin Beaudry and Pierre McKenzie. Circuits, matrices, and nonassociative computation. *Journal of Computer and System Sciences*, 50(3):441–455, 1995.
- 8 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.
- 9 S. Buss, S. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM Journal on Computing*, 21(4):755–780, 1992.
- 10 Stephan A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.
- 11 Stephen A. Cook and Lila Fontes. Formal theories for linear algebra. *Logical Methods in Computer Science*, 8(1), 2012.
- 12 Moses Ganardi, Danny Hucce, Daniel König, and Markus Lohrey. Circuit evaluation for finite semirings. Technical report, arXiv.org, 2016. <http://arxiv.org/abs/1602.04560>.
- 13 Jonathan S. Golan. *Semirings and their Applications*. Springer, 1999.

- 14 Leslie M. Goldschlager. The monotone and planar circuit value problems are log space complete for P. *SIGACT News*, 9(2):25–99, 1977.
- 15 Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, 1995.
- 16 Neil D. Jones and William T. Laaser. Complete problems for deterministic polynomial time. *Theor. Comput. Sci.*, 3(1):105–117, 1976.
- 17 Daniel König and Markus Lohrey. Evaluating matrix circuits. In *Proceedings of the 21st International Conference on Computing and Combinatorics, COCOON 2015*, volume 9198 of *Lecture Notes in Computer Science*, pages 235–248. Springer, 2015.
- 18 S. Rao Kosaraju. On parallel evaluation of classes of circuits. In *Proceedings of the 10th Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 1990*, volume 472 of *Lecture Notes in Computer Science*, pages 232–237. Springer, 1990.
- 19 Richard E. Ladner. The circuit value problem is log space complete for P. *SIGACT News*, 7(1):18–20, 1975.
- 20 Markus Lohrey. *The Compressed Word Problem for Groups*. SpringerBriefs in Mathematics. Springer, 2014.
- 21 Donald J. McCarthy and David L. Hayes. Subgroups of the power semigroup of a group. *Journal of Combinatorial Theory, Series A*, 14(2):173–186, 1973.
- 22 Pierre McKenzie and Klaus W. Wagner. The complexity of membership problems for circuits over sets of natural numbers. *Computational Complexity*, 16(3):211–244, 2007.
- 23 Gary L. Miller, Vijaya Ramachandran, and Erich Kaltofen. Efficient parallel evaluation of straight-line code and arithmetic circuits. *SIAM J. Comput.*, 17(4):687–695, 1988.
- 24 Gary L. Miller and Shang-Hua Teng. Tree-based parallel algorithm design. *Algorithmica*, 19(4):369–389, 1997. doi:10.1007/PL00009179.
- 25 Gary L. Miller and Shang-Hua Teng. The dynamic parallel complexity of computational circuits. *SIAM J. Comput.*, 28(5):1664–1688, 1999.
- 26 Cristopher Moore, Denis Thérien, François Lemieux, Joshua Berman, and Arthur Drisko. Circuits and expressions with nonassociative gates. *J. Comput. Syst. Sci.*, 60(2):368–394, 2000.
- 27 Vijaya Ramachandran and Honghua Yang. An efficient parallel algorithm for the general planar monotone circuit value problem. *SIAM J. Comput.*, 25(2):312–339, 1996.
- 28 John Rhodes and Benjamin Steinberg. *The q-theory of Finite Semigroups*. Springer, 2008.
- 29 Alexander A. Rubtsov and Mikhail N. Vyalyi. Regular realizability problems and context-free languages. In *Proceedings of the 17th International Workshop on Descriptive Complexity of Formal Systems, DCFS 2015*, volume 9118 of *Lecture Notes in Computer Science*, pages 256–267. Springer, 2015.
- 30 Stephen D. Travers. The complexity of membership problems for circuits over sets of integers. *Theor. Comput. Sci.*, 369(1-3):211–229, 2006.
- 31 Leslie G. Valiant, Sven Skyum, S. Berkowitz, and Charles Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput.*, 12(4):641–644, 1983.
- 32 Heribert Vollmer. The gap-language-technique revisited. In *Proceedings of the 4th Workshop on Computer Science Logic, CSL'90*, volume 533 of *Lecture Notes in Computer Science*, pages 389–399. Springer, 1990.
- 33 Heribert Vollmer. *Introduction to Circuit Complexity*. Springer, 1999.

# Combining Treewidth and Backdoors for CSP<sup>\*†</sup>

Robert Ganian<sup>‡1</sup>, M. S. Ramanujan<sup>2</sup>, and Stefan Szeider<sup>3</sup>

1 Algorithms and Complexity Group, TU Wien, Vienna, Austria  
rganian@ac.tuwien.ac.at

2 Algorithms and Complexity Group, TU Wien, Vienna, Austria  
ramanujan@ac.tuwien.ac.at

3 Algorithms and Complexity Group, TU Wien, Vienna, Austria  
sz@ac.tuwien.ac.at

---

## Abstract

We show that CSP is fixed-parameter tractable when parameterized by the treewidth of a backdoor into any tractable CSP problem over a finite constraint language. This result combines the two prominent approaches for achieving tractability for CSP: (i) structural restrictions on the interaction between the variables and the constraints and (ii) language restrictions on the relations that can be used inside the constraints. Apart from defining the notion of backdoor-treewidth and showing how backdoors of small treewidth can be used to efficiently solve CSP, our main technical contribution is a fixed-parameter algorithm that finds a backdoor of small treewidth.

**1998 ACM Subject Classification** G.2.1 Combinatorics, F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** Algorithms and data structures, Fixed Parameter Tractability, Constraint Satisfaction

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.36

## 1 Introduction

The Constraint Satisfaction Problem (CSP) is a central and generic computational problem which provides a common framework for many theoretical and practical applications [34]. An instance of CSP consists of a collection of variables that must be assigned values subject to constraints, where each constraint is given in terms of a relation whose tuples specify the allowed combinations of values for specified variables. The problem was originally formulated by Montanari [41], and has been found equivalent to the homomorphism problem for relational structures [22] and the problem of evaluating conjunctive queries on databases [37]. CSP is NP-complete in general, and identifying the classes of CSP instances which can be solved efficiently has become a prominent research area in theoretical computer science [10].

One of the most classical approaches in this area relies on exploiting the *structure* of how variables and constraints interact with each other, most prominently in terms of the *treewidth* of graph representations of CSP instances. The first result in this line of research dates back to 1985, when Freuder [25] observed that CSP is polynomial-time tractable if the *primal treewidth*, which is the treewidth of the *primal graph* of the instance, is bounded by a constant. A large number of related results on structural restrictions for CSP have been obtained to date (see, e.g., [13, 18, 30, 31, 40, 44]).

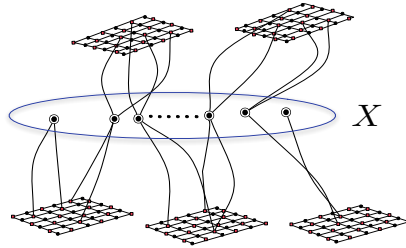
---

\* A full version of the paper is available at <https://arxiv.org/abs/1610.03298>.

† The authors acknowledge support from Austrian Science Funds (FWF), project P26696.

‡ Robert Ganian is also affiliated with FI MU, Brno, Czech Republic.





■ **Figure 1** An illustration of instances with neither a small backdoor into  $\text{CSP}(\Gamma)$  for any tractable constraint language  $\Gamma$ , nor bounded primal treewidth. Here,  $X$  denotes a minimum strong backdoor of unbounded size into  $\text{CSP}(\Gamma)$  for some choice of  $\Gamma$ . The ellipsis represents the fact that  $X$  can be arbitrarily big, implying an unbounded number of such instances, one for each size of  $X$ .

The other leading approach used to show the tractability of constraint satisfaction relies on *constraint languages*. In this case, the polynomially tractable classes are defined in terms of a tractable *constraint language*  $\Gamma$ , which is a set of relations that can be used in the instance. A landmark result in this area is Schaefer’s celebrated Dichotomy Theorem for Boolean CSP [46] which says that for every constraint language  $\Gamma$  over the Boolean domain, the corresponding CSP problem is either NP-complete or solvable in polynomial time. Feder and Vardi [22] conjectured that such a dichotomy holds for all finite constraint languages. Although the conjecture is still open it has been proven true for many important special cases (see, e.g., [6, 7, 9, 14, 17, 33]).

Tractability due to restrictions on the constraint language and tractability due to restrictions in terms of the structure of the CSP instance are often considered complementary: under structural restrictions the domain language can be arbitrary, whereas under constraint language restrictions the variables and constraints can interact arbitrarily.

One specific tool that is frequently used to build upon the constraint language approach detailed above is the notion of *backdoors*, which provides a means of relaxing celebrated results on tractable constraint languages to instances which are ‘almost’ tractable. In particular, this is done by measuring the size of a *strong backdoor* [47] to a selected tractable class, where a strong backdoor is a set of variables with the property that every assignment of these variables results in a CSP instance in the specified class. A natural way of defining such a class is to consider all CSP instances whose constraints use relations from a constraint language  $\Gamma$ , denoted by  $\text{CSP}(\Gamma)$ . The last couple of years have seen several new results for CSP using this backdoor-based approach (see, e.g., [11, 26, 27]). In particular, the general aim of research in this direction is to obtain so-called *fixed-parameter* algorithms, i.e., algorithms where the running time only has a polynomial dependence on the input size and the exponential blow-up is restricted exclusively to the size of the backdoor (the *parameter*). Parameterized decision problems which admit such an algorithm belong to the complexity class FPT.

We note that treewidth-based and backdoor-based approaches outlined above are orthogonal to each other. Consider, for example, on the one hand a CSP instance which is tractable due to the used constraint language but which has high treewidth, or on the other hand an instance consisting of many disjoint copies of CSP instances of constant primal treewidth with a constant-size strong backdoor into a tractable constraint language (backdoor size multiplies whereas treewidth remains constant). Hence applying either of these approaches (treewidth-based and backdoor-based) alone will not yield satisfactory results for instances that are not homogeneous with respect to either of these forms of restrictions. It is certainly natural to consider the algorithmic complexity of instances which have small treewidth after



certain simple ‘blocks’ characterized by a tractable constraint language are removed, or instances with a large but ‘well-structured’ backdoor to a tractable class (see Fig. 1), but until now we lacked the theoretical tools required to capture the complexity of such instances.

**Our Results.** We propose and develop a hybrid framework for constraint satisfaction which combines the advantages of both the width-based and backdoor-based approaches. In particular, we introduce the notion of *backdoor-treewidth* with respect to a constraint language  $\Gamma$ ; this is defined, roughly speaking, as the primal treewidth of the instance after contracting (possibly large) parts of the instance into single constraints, so that the remaining variables form a strong backdoor into  $\text{CSP}(\Gamma)$  in the original instance. We refer to Definition 5 for the formal definition of backdoor-treewidth. It is not difficult to see that backdoor-treewidth is at most the minimum of primal treewidth and the size of a backdoor into the specified class. However, backdoor-treewidth can be arbitrarily smaller than both the primal treewidth and the size of a backdoor, and hence promises to push the frontiers of tractability beyond the current state of the art.

► **Theorem 1.** *Let  $\Gamma$  be a fixed tractable finite constraint language. Then, CSP parameterized by the backdoor-treewidth with respect to  $\Gamma$  is FPT.*

We note that our result is in fact tight as far as the choice of the language  $\Gamma$  is concerned:  $\Gamma$  must clearly be tractable, and both the backdoor-based and width-based approaches are known to fail for infinite languages under established complexity assumptions. To be more specific, finding strong backdoors is not even FPT parameterized by backdoor size if the arity of relations in the language is unbounded [27], primal treewidth implicitly bounds the arity of relations, and both approaches require bounded domain to solve CSP in FPT time [44]. In fact, our algorithm implies that even when  $\Gamma$  is not a fixed constraint language, CSP is FPT when parameterized jointly by the maximum arity of the relations in  $\Gamma$ , the size of the domain *and* the backdoor-treewidth with respect to  $\Gamma$ .

Two separate problems need to be dealt with in order to use backdoor-treewidth for solving constraint satisfaction: finding a strong backdoor of small treewidth, and then using it to actually solve the CSP instance. The latter task can be solved efficiently by a dynamic programming procedure on a tree-decomposition. However, finding strong backdoors of small treewidth is considerably more complicated and forms the main technical contribution of this article. We note in particular that algorithms for finding small backdoors to tractable classes cannot be used for this purpose, since the size of the backdoors we are interested in can be very large. In fact, it is even far from obvious that we can detect a backdoor of treewidth at most  $k$  in polynomial time when  $k$  is considered a constant (and the order of the polynomial may depend on  $k$ ).

Our result on backdoor-treewidth also carries over to the counting variant of CSP ( $\#\text{CSP}$ ).  $\#\text{CSP}$  is a prominent  $\#\text{P}$ -complete extension of CSP problem which asks for the number of variable assignments that satisfy the given constraints. Structural restrictions as well as language restrictions have been studied for  $\#\text{CSP}$ . The dynamic programming algorithm for CSP for instances of bounded primal treewidth can be readily adapted to  $\#\text{CSP}$  (see, e.g., [21]). A constraint language  $\Gamma$  is  *$\#\text{-tractable}$*  if  $\#\text{CSP}(\Gamma)$  ( $\#\text{CSP}$  restricted to instances whose constraints use only relations from  $\Gamma$ ) can be solved in polynomial time. Bulatov [8] characterized all finite  *$\#\text{-tractable}$*  constraint languages. Applying our results, we obtain the following corollary.

► **Corollary 2.** *Let  $\Gamma$  be a fixed  $\#\text{-tractable}$  finite constraint language. Then,  $\#\text{CSP}$  parameterized by the backdoor-treewidth with respect to  $\Gamma$  is FPT.*

- (a) In the first part of our algorithm to detect strong backdoors of small treewidth, we define a notion of *boundaried* CSP instances in the spirit of boundaried graphs and show that for any  $t, k \in \mathbb{N}$ , there is an equivalence relation  $\sim_{t,k}$  on the set of all  $t$ -boundaried CSP instances such that (i) this relation has at most  $f(k, t)$  equivalence classes for some function  $f$  (all functions used in the paper can be easily seen to be computable) depending only on  $k$  and the constraint language  $\Gamma$ , and (ii) for any two  $t$ -boundaried CSP instances in the same equivalence class of  $\sim_{t,k}$ , they ‘interact in the same way’ with every other  $t$ -boundaried CSP-instance.
- (b) We then describe an algorithm that for any given  $t, k \in \mathbb{N}$  runs in time  $\mathcal{O}(g(t, k))$  for some function  $g$  and actually *constructs* a set  $\mathfrak{H}$  of  $f(k, t)$  CSP instances, one from each equivalence class of the relation  $\sim_{t,k}$ . Additionally, we show that each instance in this set has size bounded by a function of  $k$  and  $t$ .
- (c) In this part, we show that for any given  $t$ -boundaried CSP instance  $\mathbf{I}$  whose size exceeds a certain bound depending on  $k$  and  $t$  and whose incidence graph satisfies certain connectivity properties, we can in time  $g(t, k)|\mathbf{I}|^{\mathcal{O}(1)}$  correctly determine the equivalence class that this instance belongs to and compute a strictly smaller  $t$ -boundaried CSP instance  $\mathbf{I}'$  which belongs to the same equivalence class of  $\sim_{t,k}$  as  $\mathbf{I}$ . It follows that once  $\mathbf{I}'$  is computed, we can ‘replace’  $\mathbf{I}$  with the strictly smaller  $\mathbf{I}'$ , without altering the existence (or non-existence) of a strong backdoor of small treewidth. Our replacement framework is inspired by the graph replacement tools dating back to the results of Fellows and Langston [23] and further developed by Arnborg, Bodlaender, and other authors [1, 3, 5, 19, 4].
- (d) In this part, we utilize the *recursive-understanding* technique, introduced by Grohe et al. [32] to solve the Topological Subgraph Containment problem and used with great success in the design of FPT algorithms for several other fundamental graph problems (see [36, 12]), to recursively compute a  $t$ -boundaried subinstance with the properties required to execute Part (c). Once this process terminates, we have an instance whose size is upper-bounded by a function of  $k$  and  $t$  which can be solved by brute force.

**Related Work.** Williams et al. [47, 48] introduced the notion of *backdoors* for the runtime analysis of algorithms for CSP and SAT, see also [35] for a more recent discussion of backdoors for SAT. A backdoor is a small set of variables whose instantiation puts the instance into a fixed tractable class (called the base class). One distinguishes between strong and weak backdoors, where for the former all instantiations lead to an instance in the base class, and for the latter at least one leads to a satisfiable instance in the base class. Backdoors have been studied under a different name by Crama et al. [16]. The study of the parameterized complexity of finding small backdoors was initiated by Nishimura et al. [42] for SAT, who considered backdoors into the classes of Horn and Krom CNF formulas. Further results cover the classes of renamable Horn formulas [43], q-Horn formulas [28] and classes of formulas of bounded treewidth [29, 24]. The detection of backdoors for CSP has been studied in several works [2, 11]. Gaspers et al. [27] obtained results on the detection of strong backdoors into *heterogeneous* base classes of the form  $\text{CSP}(\Gamma_1) \cup \dots \cup \text{CSP}(\Gamma_d)$  where for each instantiation of the backdoor variables, the reduced instance belongs entirely to some  $\text{CSP}(\Gamma_i)$  (possibly to different  $\text{CSP}(\Gamma_i)$ ’s for different instantiations). This direction was recently further generalized by Ganian et al. [26] by developing a framework for detecting strong backdoors into so-called *scattered* base classes with respect to  $\Gamma_1 \dots \Gamma_d$ ; there, each instantiation of the backdoor variables results in a reduced instance whose every connected component belongs entirely to some  $\text{CSP}(\Gamma_i)$  (possibly to different  $\text{CSP}(\Gamma_i)$ ’s for different components and different instantiations).

## 2 Preliminaries

We use standard graph terminology, see for instance the handbook by Diestel [20]. For  $i \in \mathbb{N}$ , we use  $[i]$  to denote the set  $\{1, \dots, i\}$ .

**Constraint Satisfaction.** Let  $\mathcal{V}$  be a set of variables and  $\mathcal{D}$  a finite set of values. A *constraint of arity  $\rho$  over  $\mathcal{D}$*  is a pair  $(S, R)$  where  $S = (x_1, \dots, x_\rho)$  is a sequence of variables from  $\mathcal{V}$  and  $R \subseteq \mathcal{D}^\rho$  is a  $\rho$ -ary relation. The set  $\text{var}(C) = \{x_1, \dots, x_\rho\}$  is called the *scope* of  $C$ . An *assignment*  $\alpha : X \rightarrow \mathcal{D}$  is a mapping of a set  $X \subseteq \mathcal{V}$  of variables. An assignment  $\alpha : X \rightarrow \mathcal{D}$  *satisfies* a constraint  $C = ((x_1, \dots, x_\rho), R)$  if  $\text{var}(C) \subseteq X$  and  $(\alpha(x_1), \dots, \alpha(x_\rho)) \in R$ . For a set  $\mathbf{I}$  of constraints we write  $\text{var}(\mathbf{I}) = \bigcup_{C \in \mathbf{I}} \text{var}(C)$  and  $\text{rel}(\mathbf{I}) = \{R : (S, R) \in C, C \in \mathbf{I}\}$ .

A finite set  $\mathbf{I}$  of constraints is *satisfiable* if there exists an assignment that simultaneously satisfies all the constraints in  $\mathbf{I}$ . The *Constraint Satisfaction Problem* (CSP, for short) asks, given a finite set  $\mathbf{I}$  of constraints, whether  $\mathbf{I}$  is satisfiable. The *Counting Constraint Satisfaction Problem* (#CSP, for short) asks, given a finite set  $\mathbf{I}$  of constraints, to determine the number of assignments to  $\text{var}(\mathbf{I})$  that satisfy  $\mathbf{I}$ . CSP is NP-complete and #CSP is #P-complete (see, e.g., [8]).

Let  $\alpha : X \rightarrow \mathcal{D}$  be an assignment. For a  $\rho$ -ary constraint  $C = (S, R)$  with  $S = (x_1, \dots, x_\rho)$  and  $R \subseteq \mathcal{D}^\rho$ , we denote by  $C|_\alpha$  the constraint  $(S', R')$  obtained from  $C$  as follows.  $R'$  is obtained from  $R$  by (i) deleting all tuples  $(d_1, \dots, d_\rho)$  from  $R$  for which there is some  $1 \leq i \leq \rho$  such that  $x_i \in X$  and  $\alpha(x_i) \neq d_i$ , and (ii) removing from all remaining tuples all coordinates  $d_i$  with  $x_i \in X$ .  $S'$  is obtained from  $S$  by deleting all variables  $x_i$  with  $x_i \in X$ . For a set  $\mathbf{I}$  of constraints we define  $\mathbf{I}|_\alpha$  as  $\{C|_\alpha : C \in \mathbf{I}\}$ .

A *constraint language* (or *language*, for short)  $\Gamma$  over a domain  $\mathcal{D}$  is a set of relations (of possibly various arities) over  $\mathcal{D}$ . By  $\text{CSP}(\Gamma)$  we denote CSP restricted to instances  $\mathbf{I}$  with  $\text{rel}(\mathbf{I}) \subseteq \Gamma$ . A constraint language is *tractable* if for every finite subset  $\Gamma' \subseteq \Gamma$ , the problem  $\text{CSP}(\Gamma')$  can be solved in polynomial time. A constraint language is *#-tractable* if for every finite subset  $\Gamma' \subseteq \Gamma$ , the problem  $\#\text{CSP}(\Gamma')$  can be solved in polynomial time. Throughout this paper, we make the technical assumption that every considered tractable or #-tractable constraint language  $\Gamma$  contains the redundant tautological relation of arity 2; note that if this is not the case, then this relation can always be added into  $\Gamma$  and the resulting language will still be tractable or #-tractable, respectively. Let  $\Gamma$  be a constraint language and  $\mathbf{I}$  be an instance of CSP. A variable set  $X$  is a *strong backdoor* to  $\text{CSP}(\Gamma)$  if for each assignment  $\alpha : X \rightarrow \mathcal{D}$  it holds that  $\mathbf{I}|_\alpha \in \text{CSP}(\Gamma)$ .

The *primal graph* of a CSP instance  $\mathbf{I}$  is the graph whose vertices correspond to the variables of  $\mathbf{I}$  and where two variables  $a, b$  are adjacent iff there exists a constraint in  $\mathbf{I}$  whose scope contains both  $a$  and  $b$ . The *incidence graph* of a CSP instance  $\mathbf{I}$  is the bipartite graph whose vertices correspond to the variables and constraints of  $\mathbf{I}$ , and where vertices corresponding to a variable  $x$  and a constraint  $C$  are adjacent if and only if  $x \in \text{var}(C)$ . Observe that an incidence graph does not uniquely define a CSP instance; however, in this paper the CSP instance from which a graph is obtained will always be clear from the context. Hence for an incidence or primal graph  $G$  we will denote the corresponding CSP instance by  $\psi(G)$ . Furthermore, we slightly abuse the notation and use  $\mathcal{V}(G)$  to refer to the vertices of  $G$  that correspond to variables in  $\psi(G)$ , and  $\mathcal{C}(G)$  to refer to the vertices of  $G$  that correspond to constraints in  $\psi(G)$ . Also, for a vertex subset  $X \subseteq V(G)$ , we continue to use the notations  $\mathcal{V}(X)$  and  $\mathcal{C}(X)$  to refer to the sets  $\mathcal{V}(G) \cap X$  and  $\mathcal{C}(G) \cap X$ , respectively.

**Treewidth.** Let  $G$  be a graph. A *tree decomposition* of  $G$  is a pair  $(T, \mathcal{X} = \{X_t\}_{t \in V(T)})$  where  $T$  is a tree and  $\mathcal{X}$  is a collection of subsets of  $V(G)$  such that: (1) for each edge  $e = uv \in E(G)$  there exists some  $t \in V(T)$  such that  $\{u, v\} \subseteq X_t$ , and (2) for each vertex  $v \in V(G)$ ,  $T[\{t \mid v \in X_t\}]$  is a non-empty connected subtree of  $T$ . We call the vertices of  $T$  *nodes* and the sets in  $\mathcal{X}$  *bags* of the tree decomposition  $(T, \mathcal{X})$ . The *width* of  $(T, \mathcal{X})$  is equal to  $\max\{|X_t| - 1 \mid t \in V(T)\}$  and the *treewidth* of  $G$  (denoted  $tw(G)$ ) is the minimum width over all tree decompositions of  $G$ . The primal treewidth of a CSP instance  $\mathbf{I}$  is the treewidth of its primal graph, and similarly the incidence treewidth of  $\mathbf{I}$  is the treewidth of its incidence graph. We note that if the constraints have bounded arity, then any class of CSP instances has bounded primal treewidth if and only if it has bounded incidence treewidth [45].

► **Proposition 3** ([38]). *Let  $\mathbf{I}$  be a CSP instance where the constraints have arity bounded by  $\rho \in \mathbb{N}$ . Then, the primal treewidth of the instance is at most  $\rho(t + 1) - 1$  where  $t$  is the incidence treewidth of the instance.*

**$t$ -Boundaried CSP Instances.** A  *$t$ -boundaried graph* is a graph  $G$  with a set  $Z \subseteq V(G)$  of size at most  $t$  with each vertex  $v \in Z$  having a unique label  $\ell(v) \in \{1, \dots, t\}$ . We refer to  $Z$  as the *boundary* of  $G$ . For a  $t$ -boundaried graph  $G$ ,  $\delta(G)$  denotes the boundary of  $G$ . When it is clear from the context, we will often use the notation  $(G, Z)$  to refer to a  $t$ -boundaried graph  $G$  with boundary  $Z$ . For  $P \subseteq [t]$ , we use  $P(G, Z)$  to denote the subset of  $Z$  with labels in  $P$ ; for  $i \in [t]$  we use  $i(G, Z)$  instead of  $\{i\}(G, Z)$  for brevity. Two  $t$ -boundaried graphs  $G_1$  and  $G_2$  can be ‘glued’ together to obtain a new graph, which we denote by  $G_1 \oplus G_2$ . 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.

In some cases, we will also use a natural notion of replacement of boundaried graphs. Let  $(G_1, Z_1)$  be a  $t$ -boundaried graph which is an induced subgraph of a graph  $G$  such that  $Z_1$  is a separator between  $V(G_1) \setminus Z_1$  and  $V(G) \setminus V(G_1)$ . Let  $(G_2, Z_2)$  be a  $t$ -boundaried graph. Then the operation of *replacement* of  $(G_1, Z_1)$  by  $(G_2, Z_2)$  results in the graph  $G' = (G[V(G) \setminus (V(G_1) \setminus Z_1)], Z_1 \oplus (G_2, Z_2))$ . Furthermore, if  $G$  was a  $j$ -boundaried graph with boundary  $Z$  and  $Z \cap V(G_1) \subseteq Z_1$ , then the resulting graph  $G'$  is also a  $j$ -boundaried graph with the same boundary.

In this paper, it will sometimes be useful to lift the notions of boundaries and gluing from graphs to CSP instances. A  *$t$ -boundaried incidence graph* of a CSP instance  $\mathbf{I}$  is a  $t$ -boundaried graph  $G$  with boundary  $Z$  such that  $G$  is the incidence graph of  $\mathbf{I}$  and  $Z \subseteq \mathcal{V}$ . Similarly, we call a CSP instance  $\mathbf{I}$  with  $t$  uniquely labeled variables a  *$t$ -boundaried CSP instance*. Note that boundaried incidence graphs and boundaried CSP instances are de-facto interchangeable, but in some cases it is easier to use one rather than the other due to technical reasons.

The gluing operations of boundaried incidence graphs and boundaried CSP instances are defined analogously as for standard boundaried graphs. Observe that if  $G_1$  and  $G_2$  are  $t$ -boundaried incidence graphs of  $\mathbf{I}_1$  and  $\mathbf{I}_2$ , respectively, then  $G_1 \oplus G_2$  is also an incidence graph; furthermore,  $\psi(G_1 \oplus G_2)$  is well-defined and can be reconstructed from  $\mathbf{I}_1$  and  $\mathbf{I}_2$ .

### 3 Backdoor-Treewidth

In this section we give a formal definition of the notion of backdoor-treewidth.

► **Definition 4.** Let  $G$  be a graph and  $X \subseteq V(G)$ . We denote by  $\mathbf{Torso}_G(X)$  the following graph defined over the vertex set  $X$ . For every pair of vertices  $x_1, x_2 \in X$ , we add the edge

$(x_1, x_2)$  if (a)  $(x_1, x_2) \in E(G)$  or (b)  $x_1$  and  $x_2$  both have a neighbor in the same connected component of  $G - X$ . That is, we begin with  $G[X]$  and make the neighborhood of every connected component of  $G - X$ , a clique. When  $G$  is an incidence graph of the instance  $\mathbf{I}$  and  $X$  is a set of variables of  $\mathbf{I}$ , we also refer to  $\mathbf{Torso}_G(X)$  as  $\mathbf{Torso}_{\mathbf{I}}(X)$ .

► **Definition 5.** Let  $\mathcal{F}$  be a class of CSP instances and  $\mathbf{I}$  be a CSP instance. Then the *backdoor-treewidth* of  $\mathbf{I}$  with respect to  $\mathcal{F}$ , denoted  $\mathbf{btw}_{\mathcal{F}}(\mathbf{I})$ , is the smallest value of  $tw(\mathbf{Torso}_{\mathbf{I}}(X))$  taken over all strong backdoors  $X$  of  $\mathbf{I}$  into  $\mathcal{F}$ . If  $\mathcal{F} = \text{CSP}(\Gamma)$  for some constraint language  $\Gamma$ , then we call  $\mathbf{btw}_{\mathcal{F}}$  the backdoor-treewidth with respect to  $\Gamma$ .

As an example, observe that in Figure 1 the graph  $\mathbf{Torso}_G(X)$  is a path. Throughout this paper, we sometimes refer to backdoors of small treewidth simply as backdoors of small *width*. Next, we show how backdoors of small treewidth can be used to solve CSP and  $\#\text{CSP}$ .

► **Lemma 6.** *Let  $\mathbf{I}$  be a CSP instance over domain  $\mathcal{D}$  and  $X$  be a strong backdoor of  $\mathbf{I}$  to the class  $\mathcal{F}$ . There is an algorithm that, given  $\mathbf{I}$  and  $X$ , runs in time  $\mathcal{O}(|\mathcal{D}|^{tw(\mathbf{Torso}(X))} |\mathbf{I}|^{\mathcal{O}(1)})$  and correctly decides whether  $\mathbf{I}$  is satisfiable or not. Furthermore, if  $\mathcal{F}$  is  $\#\text{-tractable}$  and  $X$  is a strong backdoor to  $\mathcal{F}$ , then in the same time bound one can count the number of satisfying assignments of  $\mathbf{I}$ .*

**Sketch of Proof.** The algorithm is a standard dynamic programming procedure over a bounded treewidth graph and hence we only sketch it briefly. Let  $G$  denote the incidence graph of  $\mathbf{I}$  and let  $H$  denote the graph  $\mathbf{Torso}(X)$  and let  $(T, \mathcal{X})$  be a tree-decomposition of  $H$  of width  $tw(H)$ . Now, for every  $v \in T$ , we define the instance  $\mathbf{I}_v$  as the subinstance of  $\mathbf{I}$  induced on the variables in  $X_v$ , the bags below it in  $(T, \mathcal{X})$ , and the constraints whose scope is completely contained in the union of  $X_v$  and the bags below it. The key observation is that for any connected component of  $G - X$ , there is a vertex  $v \in V(T)$  such that the bag  $X_v$  contains the neighbors of this component.

To solve CSP, for each  $v \in T$  we define a function  $\tau_v$  which maps assignments of the variables in  $X_v$  to 0 to 1. Let  $\gamma : X_v \rightarrow \mathcal{D}$  be an assignment to the variables in  $X_v$ . We define  $\tau_v(\gamma) = 1$  if there is a satisfying assignment for  $\mathbf{I}_v$  that extends  $\gamma$  and  $\tau_v(\gamma) = 0$  otherwise. Let  $v^*$  denote the root of  $T$ . Clearly, the instance  $\mathbf{I}$  is satisfiable if and only if there is a  $\gamma : X_{v^*} \rightarrow \mathcal{D}$  such that  $\tau_{v^*}(\gamma) = 1$ . At this point it suffices to describe how to dynamically compute the function  $\tau_v$  for each node in the tree-decomposition; this step can be facilitated by the use of so-called nice tree-decompositions. The algorithm to solve  $\#\text{CSP}$  is similar; there  $\tau_v$  is extended by information about how many ways there are to extend an assignment to the variables in  $X_v$  to a satisfying assignment for  $\mathbf{I}_v$ . ◀

As the width of a backdoor can be arbitrarily smaller than its size, the width provides a much better measure of how far away an instance is from a tractable base class. In particular, the width lower-bounds both the primal treewidth and the backdoor size. We formalize this below.

► **Proposition 7.** *Let  $\mathbf{I}$  be a CSP instance and  $\mathcal{F}$  be a class of CSP instances. Let  $q$  be the primal treewidth of  $\mathbf{I}$  and  $r$  be the minimum size of a strong backdoor to  $\mathcal{F}$  in  $\mathbf{I}$ . Then  $\mathbf{btw}_{\mathcal{F}}(\mathbf{I}) \leq \min(q, r)$ .*

In order to prove Theorem 1, we give an FPT algorithm for the problem of finding strong backdoors parameterized by their width (formalized below). We note that since we state our results in as general terms as possible, the dependence on  $k$  is likely to be sub-optimal for specific languages and could be improved using properties specific to each language.

WIDTH STRONG-CSP( $\Gamma$ ) BACKDOOR DETECTION

Parameter:  $k$

**Input:** CSP instance  $\mathbf{I}$ , integer  $k$ .

**Objective:** Return a set  $X$  of variables such that  $X$  is a strong backdoor of  $\mathbf{I}$  to CSP( $\Gamma$ ) of width at most  $k$  or correctly conclude that no such set exists.

The main technical content of the article then lies in the proof of the following theorem.

► **Theorem 8.** WIDTH STRONG-CSP( $\Gamma$ ) BACKDOOR DETECTION is FPT for every finite  $\Gamma$ .

Before we proceed to the description of the algorithms, we state the following simple and obvious preprocessing routine (correctness is argued in the full version of this paper, available at <https://arxiv.org/abs/1610.03298>) which will allow us to infer certain structural information regarding interesting instances of this problem.

► **Reduction Rule 9.** Given a CSP instance  $\mathbf{I}$  and an integer  $k$  as an instance of WIDTH STRONG-CSP( $\Gamma$ ) BACKDOOR DETECTION, if there is a constraint in  $\mathbf{I}$  of arity at least  $p + k + 2$  where  $p$  is the maximum arity of a relation in  $\Gamma$ , then return NO.

## 4 The Finite State Lemma

In this section, we prove that the problem WIDTH STRONG-CSP( $\Gamma$ ) BACKDOOR DETECTION has finite state; this will allow us to construct a finite set of bounded-size representatives (Section 5) which will play a crucial role in the proof of Theorem 8 (Section 6). Let  $\Gamma$  be a finite constraint language; throughout the rest of the paper, we work with this fixed constraint language. We begin by defining a relation over the set of boundaried incidence graphs.

► **Definition 10.** Let  $k, t \in \mathbb{N}$  and let  $(G_1, Z_1)$  and  $(G_2, Z_2)$  be  $t$ -boundaried incidence graphs of CSP instances  $\mathbf{I}_1$  and  $\mathbf{I}_2$  with boundaries  $Z_1$  and  $Z_2$  respectively. Then, we say that  $(\mathbf{I}_1, Z_1) \sim_{t,k} (\mathbf{I}_2, Z_2)$  (or  $(G_1, Z_1) \sim_{t,k} (G_2, Z_2)$ ) if for every  $t$ -boundaried CSP instance  $\mathbf{I}_3$  with incidence graph  $G_3$ , the instance  $\psi(G_1 \oplus G_3)$  has a strong backdoor set of width at most  $k$  into CSP( $\Gamma$ ) if and only if the instance  $\psi(G_2 \oplus G_3)$  has a strong backdoor set of width at most  $k$  into CSP( $\Gamma$ ).

It is clear that  $\sim_{t,k}$  is an equivalence relation. Generally speaking, the high-level goal of this section is to prove that  $\sim_{t,k}$  has finite index. This is achieved by introducing a second, more technical equivalence  $\equiv_{t,h,\varepsilon}$  which captures all the information about how a  $t$ -boundaried incidence graph  $(G, Z)$  contributes to the (non)-existence of a strong backdoor of small width after gluing. Observe that for a set  $X$  which has vertices from ‘both’ sides of a boundary the graph  $\mathbf{Torso}(X)$  may have edges crossing this boundary. Since we need to take this behaviour into account, proving this lemma is in fact much more involved than might be expected at first glance.

To define  $\equiv_{t,h,\varepsilon}$ , we will first need the notion of a *configuration*, which can be thought of as one possible way a  $t$ -boundaried graph can interact via gluing; this is then tied to the notion of a *realizable configuration*, which is a configuration that actually can occur in the graph  $(G, Z)$ . We let  $(G_1, Z_1) \equiv_{t,h,\varepsilon} (G_2, Z_2)$  if and only if both boundaried graphs have the same set of realizable configurations. Before we proceed to the technical definition of a configuration, we need one more bit of notation. Since we will often be dealing with labeled minors, we fix a pair of symbols  $\square$  and  $\diamond$  and express all relevant label sets using these symbols. Specifically, for  $r, s \in \mathbb{N}$  we let  $\mathcal{L}(r, s)$  denote the set  $2^{\{\square_1, \dots, \square_r\}} \cup \{\diamond_1, \dots, \diamond_s\}$ .

► **Definition 11.** Let  $h, t \in \mathbb{N}$ . A  $(t, h)$ -**configuration** is a tuple  $(P, w, w', \mathcal{P}, \mathcal{P}', \gamma, \mathfrak{H})$ , where:

1.  $P$  is a subset of  $[t]$ ,
2.  $w, w' \in \mathbb{N}$  and  $w' \leq (w + 1)t$ ,
3.  $\mathcal{P} = \{Q_1, \dots, Q_r\}$  is a partition of  $[t] \setminus P$ ,
4.  $\mathcal{P}' \in 2^{\binom{P}{2}} \times 2^{\binom{[w']}{2}} \times 2^{P \times [w']}$ ,
5.  $\gamma: \mathcal{P} \rightarrow 2^P \times 2^{[w']}$ ,
6.  $\mathfrak{H}$  is a collection of labeled graphs on at most  $h$  vertices where the label set is  $\mathfrak{L}(t, w')$ .

For a set  $Q \in \mathcal{P}$  with  $\gamma(Q) = (J_1, J_2)$ , we denote by  $\gamma^i(Q)$  the set  $J_i$  for each  $i \in \{1, 2\}$ . A  $(t, h)$ -configuration  $(P, w, w', \mathcal{P}, \mathcal{P}', \gamma, \mathfrak{H})$  is called a  $(t, h, \varepsilon)$ -**configuration** if  $w \leq \varepsilon$  and we denote the set of such  $(t, h)$ -configurations by  $\mathfrak{C}(t, h, \varepsilon)$ .

Let us informally break down the intuition behind the above definition.  $t$  corresponds to the size of the boundary of the associated  $t$ -boundaried incidence graph (as we will see in the next definition), and  $h$  is an upper bound on the size of forbidden minors for our target treewidth. The  $(t, h)$ -configuration then captures the following information about interactions between a  $t$ -boundaried incidence graph  $(G_1, Z_1)$  and a potential solution after gluing:

- (a)  $P$  represents the part of the boundary that intersects a backdoor of small width,
- (b)  $w'$  represents neighbors of the remainder of the boundary outside of  $G_1$ ,
- (c)  $w$  represents the target treewidth of the torso,
- (d)  $\mathcal{P}$  represents how the part of the boundary outside of the strong backdoor will be partitioned into connected components, i.e., how it will ‘collapse’ into the torso,
- (e)  $\mathcal{P}'$  represents all the new edges that will be created in the torso due to collapsing of parts outside of the torso,
- (f)  $\gamma$  represents connections between connected components in the boundary outside of the strong backdoor and relevant variables in the strong backdoor, which is the second part of information needed to encode the collapse of these components into the torso,
- (g)  $\mathfrak{H}$  represents ‘parts’ of all minors of size at most  $h$  present in the torso inside  $G_1$ .

In order to formally capture the intuition outlined above, we define the result of ‘applying’ a configuration on a  $t$ -boundaried incidence graph.

► **Definition 12.** Let  $h, t \in \mathbb{N}$ ,  $(G, Z)$  be the  $t$ -boundaried incidence graph of a  $t$ -boundaried CSP instance  $\mathbf{I}$  and  $\omega = (P, w, w', \mathcal{P}, \mathcal{P}', \gamma, \mathfrak{H})$  be a  $(t, h)$ -configuration. We associate with  $G$  and  $\omega$  an incidence graph  $G^\omega$  which is defined as follows. We begin with the graph  $G$ , add  $w'$  new variables  $l_1^\omega, \dots, l_{w'}^\omega$ , denoting the set comprising these vertices by  $L_\omega$ . For every  $J \subseteq [w']$ , we denote by  $J(L_\omega)$  the set  $\{l_i^\omega \mid i \in J\}$ . For each  $Q \in \mathcal{P}$ , let  $(J_1^Q, J_2^Q) = \gamma(Q)$  and add  $|Q| - 1$  redundant binary constraints  $C_1^Q, \dots, C_{|Q|-1}^Q$  (we have assumed that  $\Gamma$  also contains a tautological relation of arity 2) and connect these with the variables in  $Q(G, Z)$  to form a path which alternates between a vertex/variable in  $Q(G, Z)$  and a vertex/variable in  $\{C_1^Q, \dots, C_{|Q|-1}^Q\}$ . Following this, for every variable  $u$  in  $J_1(G, Z) \cup J_2(L_\omega)$ , we add a redundant binary constraint  $C_u$  and set  $\text{var}(C_u)$  as  $u$  and an arbitrary variable in  $Q(G, Z)$ . This completes the definition of  $G^\omega$ . We also define the graph  $\tilde{G}^\omega$  as the graph obtained from  $G^\omega$  by doing the following. Let  $\mathcal{P}' = (X_1, X_2, X_3)$  where  $X_1 \subseteq \binom{P}{2}$ ,  $X_2 \subseteq \binom{[w']}{2}$  and  $X_3 \subseteq P \times [w']$ . For every pair  $(i, j) \in X_1$ , we add the edge  $(i(G, Z), j(G, Z))$ . Similarly, for every pair  $(i, j) \in X_2$ , we add the edge  $(l_i^\omega, l_j^\omega)$ . Finally, for every pair  $(i, j) \in X_3$ , we add the edge  $(i(G, Z), l_j^\omega)$ . This completes the description of  $\tilde{G}^\omega$ .

The graph  $G^\omega$  defined above can be seen as an enrichment of  $G$  by (1) adding strong backdoor variables which will be affected by a collapse of the boundary into the torso

$(l_1^\omega, \dots, l_{w'}^\omega)$  and (2) enforcing the assumed partition of part of the boundary into connected components (as per  $\mathcal{P}$ ) and (3) adding connections of these components both into the rest of the boundary and vertices  $l_i^\omega$  (as per  $\gamma$ ). The graph  $\tilde{G}^\omega$  is then an extension of  $G^\omega$  by edges which will be created in the torso. Note that while  $G^\omega$  is an incidence graph,  $\tilde{G}^\omega$  is not necessarily a bipartite graph.

With  $\tilde{G}^\omega$  in hand, we can finally formally determine whether the information contained in a given configuration is of any relevance for the given graph. This is achieved via the notion of *realizability*.

► **Definition 13.** Let  $h, t \in \mathbb{N}$ ,  $(G, Z)$  be the  $t$ -boundaried incidence graph corresponding to a  $t$ -boundaried CSP instance  $\mathbf{I}$  and let  $\omega = (P, w, w', \mathcal{P}, \mathcal{P}', \gamma, \mathfrak{H})$  be a  $(t, h)$ -configuration. We say that  $\omega$  is a **realizable** configuration in  $(G, Z)$  if, and only if, there is a subset  $S^* \subseteq \mathcal{V}(G)$  with the following properties:

1.  $S^* \cap Z = P(G, Z)$ ,
2.  $tw(\mathbf{Torso}_{\tilde{G}^\omega}(S^* \cup L_\omega))$  is at most  $w$ ,
3.  $\mathfrak{H}$  is precisely the set of all labeled minors of  $(\mathbf{Torso}_{\tilde{G}^\omega}(S^* \cup L_\omega), \Lambda_\omega)$  with at most  $h$  vertices,
4.  $S^*$  is a strong backdoor of  $\psi(G)$  into  $\text{CSP}(\Gamma)$ .

If the above conditions hold, we say that  $S^*$  **realizes**  $\omega$  in  $(G, Z)$ .

We let  $\mathfrak{S}((G, Z), h, \varepsilon)$  denote the set of all realizable  $(|Z|, h, \varepsilon)$ -configurations in  $(G, Z)$ . We ignore the explicit reference to  $Z$  in the notation if it is clear from the context. We let  $h^*(k)$  denote the upper bound on the size of forbidden minors for graphs of treewidth at most  $k$  given in [39]. For technical reasons, we will be in fact concerned with minors of size slightly greater than  $h^*(k)$ , and hence for  $t \in \mathbb{N}$  we set  $h^*(k, t) = h^*(k) + t \cdot (k + 1)$ .

We use  $\Upsilon(t, h, \varepsilon)$  to denote a computable upper bound on the number of  $(t, h, \varepsilon)$ -configurations. Observe that setting  $\Upsilon(t, h, \varepsilon) = 2^t \cdot \varepsilon \cdot \varepsilon t \cdot t^t \cdot 2^{\binom{t}{2}} \cdot 2^{\binom{\varepsilon+1}{2}t} \cdot 2^{t^2(\varepsilon+1)} \cdot 2^{t^2(\varepsilon+1)} \cdot 2^{\binom{h}{2}} h^{2^{(\varepsilon+1)t}}$  is sufficient. We now give the formal definition of the refined equivalence relation.

► **Definition 14.** Let  $t, h \in \mathbb{N}$  and let  $(\mathbf{I}_1, Z_1)$  and  $(\mathbf{I}_2, Z_2)$  be  $t$ -boundaried CSP instances with  $t$ -boundaried incidence graphs  $(G_1, Z_1)$  and  $(G_2, Z_2)$  respectively. Then,  $(\mathbf{I}_1, Z_1) \equiv_{t, h, \varepsilon} (\mathbf{I}_2, Z_2)$  (or  $(G_1, Z_1) \equiv_{t, h, \varepsilon} (G_2, Z_2)$ ) if  $\mathfrak{S}((G_1, Z_1), h, \varepsilon) = \mathfrak{S}((G_2, Z_2), h, \varepsilon)$ .

From these definitions, it is straightforward to verify that  $\equiv_{t, h, \varepsilon}$  is indeed an equivalence and the number of equivalence classes induced by this relation over the set of all  $t$ -boundaried incidence graphs is at most  $2^{\Upsilon(t, h, \varepsilon)}$ . The main lemma of this section, Lemma 15, then links  $\equiv_{t, h, \varepsilon}$  to  $\sim_{t, k}$ , and in particular shows that the former is a refinement of the latter. We note that the more refined  $\equiv_{t, h, \varepsilon}$  is used throughout the paper; it is not merely a tool for showing finite-stateness of  $\sim_{t, k}$ .

► **Lemma 15.** Let  $k, t \in \mathbb{N}$  and let  $(G_1, Z_1), (G_2, Z_2)$  be two  $t$ -boundaried incidence graphs satisfying  $(G_1, Z_1) \equiv_{t, h^*(k, t), k} (G_2, Z_2)$ . Then,  $(G_1, Z_1) \sim_{t, k} (G_2, Z_2)$ .

## 5 Computing a Bound on the Size of a Minimal Representative of $\sim_{t, k}$

In this section, we define a function  $\alpha$  such that for every  $t, k \in \mathbb{N}$ , every equivalence class of  $\sim_{t, k}$  contains a boundaried incidence graph whose size is bounded by  $\alpha(t, k)$ . In order to do so, we use the fact the relation  $\equiv_{t, h^*(k, t), k}$  refines  $\sim_{t, k}$ . The following is a brief sketch of the proof strategy.



- In the first step, we show that for any  $t$ -boundaried incidence graph  $(G, Z)$  whose *treewidth* is bounded as a function of  $t$  and  $k$  and size exceeds a certain bound also depending only on  $t$  and  $k$ , there is a strictly smaller  $t$ -boundaried graph  $(G', Z')$  such that  $(G, Z) \equiv_{t, h^*(k, t), k} (G', Z')$ . This in turn implies that for any  $t$ -boundaried incidence graph  $(G, Z)$  whose *treewidth* is bounded by a function of  $t$  and  $k$  there is a  $t$ -boundaried graph  $(G', Z')$  such that  $(G, Z) \equiv_{t, h^*(k, t), k} (G', Z')$  and the size of  $G'$  is bounded by a function of  $t$  and  $k$ .
- In the second step, we show that for any  $t$ -boundaried incidence graph  $(G, Z)$ , there is a  $t$ -boundaried incidence graph  $(G', Z')$  such that  $G'$  has *treewidth* bounded by a function of  $k$  and  $t$  and  $(G, Z) \sim_{t, k} (G', Z')$ . Combining these two steps, we obtain the following lemma.

► **Lemma 16.** *Let  $k, t \in \mathbb{N}$ . There exists a computable function  $\eta(t, k)$  and a set  $\mathfrak{F}_s(t, h^*(k, t), k)$  of at most  $\eta(t, k)$   $t$ -boundaried CSP instances that contains a  $t$ -boundaried CSP instance from every equivalence class of  $\sim_{t, k}$ . Furthermore, given  $k$  and  $t$ , the set  $\mathfrak{F}_s(t, h^*(k, t), k)$  can be computed in time  $\mathcal{O}(|\mathfrak{F}_s(t, h^*(k, t), k)|)$ .*

## 6 The FPT Algorithm for Width Strong-CSP( $\Gamma$ ) Backdoor Detection

An often-used approach in the design of FPT algorithms for graph problems is that of finding a sufficiently small separator in the graph and then reducing one of the sides. In the technique of ‘recursive understanding’ introduced by Grohe et al. [32], this is achieved by performing this step *recursively* until we arrive at a separator where the side we want to reduce has certain connectivity-based structure using which we can find a way reduce it without recursing further. This approach has been combined with various problem specific reduction rules at the bottom to obtain parameterized algorithms for several well-studied problems. These include the  $k$ -WAY CUT problem, solved by Kawarabayashi and Thorup [36], STEINER CUT and UNIQUE LABEL COVER – both solved by Chitnis et al. [12]. In this section, we will employ this technique to design our algorithm for WIDTH STRONG-CSP( $\Gamma$ ) BACKDOOR DETECTION. We begin by defining a notion of *nice* instances which basically capture the kind of instances we will be dealing with at the bottom of our recursion.

### 6.1 Solving Nice Instances

► **Lemma 17.** *There is a function  $\mathfrak{Z} : \mathbb{N}^2 \rightarrow \mathbb{N}$  and an algorithm that, given a CSP instance  $\mathbf{I}$  with incidence graph  $G$  and positive integers  $\beta, k \in \mathbb{N}$ , runs in time  $\mathcal{O}(\mathfrak{Z}(\beta, k)|G|^2)$  and either computes a strong backdoor into CSP( $\Gamma$ ) of width at most  $k$  or correctly concludes that  $\mathbf{I}$  has no backdoor set  $X$  of width at most  $k$  that satisfies the following properties:*

1.  $G - X$  has exactly one connected component  $C$  of size at least  $\beta + 1$ .
2.  $|V(G) \setminus N[C]| \leq \beta$

We now give the definition of ‘nice’ instances. Generally speaking, these are instances which fall into either the bounded ‘classical’ *treewidth* case or bounded backdoor size case.

► **Definition 18.** Let  $k, \beta \in \mathbb{N}$  and  $\mathbf{I}$  be a CSP instance with incidence graph  $G$ . We say that  $\mathbf{I}$  is  $(\beta, k)$ -*nice* if  $tw(G) \leq \beta + k$  or if  $\mathbf{I}$  has some strong backdoor set of width at most  $k$ , then it also has a strong backdoor set  $X$  of width at most  $k$  such that  $G - X$  has exactly one connected component  $C$  of size at least  $\beta + 1$ , and  $|V(G) \setminus N[C]| \leq \beta$ .

We now formally show that given a  $(\beta, k)$ -nice incidence graph, one can detect strong backdoor sets of small width in FPT time parameterized by  $\beta + k$ . This will later be used to compute small representatives of large boundaried CSP instances (specifically, in Lemma 23).

► **Lemma 19.** *There is a function  $\hat{\mathfrak{X}} : \mathbb{N} \rightarrow \mathbb{N}$  and an algorithm that, given  $\beta, k \in \mathbb{N}$ , a  $(\beta, k)$ -nice CSP instance  $\mathbf{I}$  with the incidence graph  $G$ , runs in time  $\mathcal{O}(\hat{\mathfrak{X}}(\beta + k)|G|^2)$  and either computes a strong backdoor set into  $\text{CSP}(\Gamma)$  of width at most  $k$  or correctly detects that such a set does not exist.*

**Proof.** If  $\text{tw}(G) \leq \beta + k$ , then we can solve the problem directly by applying Courcelle's Theorem [15], as follows. First, recall that the arity of any constraint which appears in the CSP instance  $\psi(G)$  is upper-bounded by  $k$  plus the maximum arity of relations in  $\Gamma$ . Hence we can assume that the number of relations which appear in the constraints of  $\psi(G)$  is bounded by a function of  $k$ , and we can think of  $G$  as having vertex labels which specify which relation is used in each constraint vertex and edge labels which specify the order in which variables appear in the incident constraint. Second, for a  $j$ -ary relation  $R$  which appears in a constraint  $C$  in  $\psi(G)$ , we say that a subset  $\alpha$  of  $\{1, \dots, j\}$  is a *valid choice* if the variables occurring in positions  $\alpha$  in  $C$  form a strong backdoor for  $\{C\}$  into  $\text{CSP}(\Gamma)$ . Note that the set of valid choices for all of the relations which occur in a constraint in  $\psi(G)$  can be precomputed in advance. Then the problem can be formulated in Monadic Second Order logic with a sentence stating the following: there exists a set  $T$  of variables such that (1) for each constraint  $C$  with label  $R$  it holds that the edges between  $T$  and  $C$  correspond to a valid choice for  $R$ , and (2) the torso of  $T$  does not contain any of the forbidden minors for treewidth at most  $k$ . Indeed, condition (1) ensures that  $T$  forms a backdoor to  $\text{CSP}(\Gamma)$  and condition (2) ensures that  $T$  has width at most  $k$ .

Otherwise, we execute the algorithm of Lemma 17 that runs in time  $\mathcal{O}(\hat{\mathfrak{Z}}(\alpha)|G|^2)$ . The function  $\mathfrak{X}$  is obtained from the function  $\mathfrak{Z}$  and the dependence of the algorithm on  $\beta + k$  in the case of bounded treewidth. ◀

## 6.2 Computing a Minimal Representative

In this subsection, we show that if a  $t$ -boundaried instance has a certain guarantee on the (non-)existence of a small separator separating two large parts of the instance from each other, then we can compute a  $t$ -boundaried instance of bounded size which is equivalent to it under the relation  $\sim_{t,k}$ .

► **Definition 20.** Let  $G$  be an incidence graph and  $(A, S, B)$  be a partition of  $V(G)$  where  $S \subseteq \mathcal{V}(G)$  and  $N(A), N(B) \subseteq S$ . We call  $(A, S, B)$  a  $(q, k)$ -*separation* if  $S$  has size at most  $k$ , and  $A$  and  $B$  have size at least  $q$ .

► **Lemma 21.** *Let  $G$  be the incidence graph of a CSP instance  $\mathbf{I}$ . If  $G$  has no  $(q, k + 1)$ -separation then  $\mathbf{I}$  is  $(q, k)$ -nice.*

► **Lemma 22.** *Let  $t \in \mathbb{N}$  and  $\mathbf{I}_1$  be a  $t$ -boundaried CSP instance with  $t$ -boundaried incidence graph  $(G, Z)$ . Let  $k, q \in \mathbb{N}$  be such that  $G$  does not admit a  $(8^q, k + 1)$ -separation, and let  $(H, J)$  be the  $t$ -boundaried incidence graph of a  $t$ -boundaried CSP instance  $\mathbf{I}_2$  such that the size of  $V(H)$  is at most some  $r \in \mathbb{N}$ . Then the incidence graph  $G \oplus H$  corresponding to the instance  $\mathbf{I}_1 \oplus \mathbf{I}_2$  has no  $(8^q + r, k + 1)$ -separation.*

For the following lemma, recall the definition of the set  $\mathfrak{F}_s(t, h^*(k, t), k)$  (Lemma 16). The proof relies on Lemmas 22, 21, and 19.

► **Lemma 23.** *Let  $t \in \mathbb{N}$  and  $\mathbf{I}_1$  be a  $t$ -boundaried CSP instance with incidence graph  $G$  and boundary  $Z$ . Further, let  $k, q \in \mathbb{N}$  be such that  $t \leq 2(k + 1)$ ,  $|V(G)| > q$ , and  $G$  does not admit a  $(8^q, k + 1)$ -separation. Let  $(H, J)$  be the  $t$ -boundaried incidence graph of a  $t$ -boundaried*

CSP instance  $\mathbf{I}_2$  in  $\mathfrak{F}_s(t, h^*(k, t), k)$ . Then the instance  $\mathbf{I}_1 \oplus \mathbf{I}_2$  is  $(8^q + \alpha(k, 2(k+1)), k)$ -nice. Furthermore, if  $q = \alpha(k, 2(k+1))$  then one can compute in time  $\mathcal{O}(\mathfrak{M}(k)|G|^2)$  a  $t$ -boundaried CSP instance  $\mathbf{I}_1^*$  of size at most  $q$  such that  $\mathbf{I}_1 \sim_{t,k} \mathbf{I}_1^*$ , for some function  $\mathfrak{M}$ .

### 6.3 Solving the Problem via Recursive Understanding

In this subsection, we complete our algorithm for WIDTH STRONG-CSP( $\Gamma$ ) BACKDOOR DETECTION by describing the recursive phase of our algorithm and the way we utilize the subroutines described earlier to solve the problem. We note that variants of Lemma 24, Lemma 25 and Lemma 27 are well-known in literature (see for example [12]). However the parameters involved in these lemmas are specific to the application. Furthermore, our proofs are simpler and avoid the color coding technique employed in [12]. Following that, we use Lemma 24 to obtain the final ingredient for our algorithm.

► **Lemma 24.** *There is an algorithm that, given an incidence graph  $G$  and  $q, k \in \mathbb{N}$ , runs in time  $\mathcal{O}((2q)^k \cdot |G|^2)$  and either computes a  $(q, k)$ -separation or concludes correctly that there is no  $(q, k)$ -separation  $(A, S, B)$  where  $A$  and  $B$  are connected.*

► **Lemma 25.** *There is an algorithm that, given an incidence graph  $G$  and  $q, k \in \mathbb{N}$ , runs in time  $\mathcal{O}((q+k)^k |G|^2)$  and either computes a  $(q, k)$ -separation in  $G$  or correctly concludes that  $G$  has no  $(8^q, k)$ -separation.*

► **Observation 26.** *Let  $(G, Z)$  be a  $t$ -boundaried graph with  $|V(G)| > q$  and  $t \leq 2(k+1)$  and let  $(A, S, B)$  be a  $(q, k+1)$ -separation in  $G$ . Then, one of the pairs  $(G[A \cup S], S \cup (Z \cap A))$  or  $(G[B \cup S], S \cup (Z \cap B))$  is a  $t'$ -boundaried graph with  $t' \leq 2(k+1)$ .*

► **Lemma 27.** *There is an algorithm that, given a  $t$ -boundaried graph  $(G, Z)$  with  $|V(G)| > q$  and  $t \leq 2(k+1)$ , in time  $\mathcal{O}((q+k)^k |G|^3)$  returns a  $t'$ -boundaried graph  $(G', Z')$  where  $G'$  is a subgraph of  $G$ ,*

- (a)  $|V(G')| > q$ ,
- (b)  $t' \leq 2k+1$ , and
- (c)  $G'$  has no  $(8^q, k+1)$ -separation.

**Proof.** We begin by executing the algorithm of Lemma 25. If this algorithm returns that  $G$  has no  $(8^q, k+1)$ -separation then we terminate the algorithm and return the graph  $(G, Z)$  itself. Otherwise, let  $(X, S, Y)$  be the  $(q, k+1)$ -separation returned by this algorithm. By Observation 26, we may assume w.l.o.g. that  $(G[X \cup S], S \cup (Z \cap X))$  is a  $t''$ -boundaried graph where  $t'' \leq 2(k+1)$ . We now set  $G := G[X \cup S]$ ,  $Z := S \cup (Z \cap X)$  and recurse. Since the depth of recursion is bounded by the size of the input graph and each step takes time  $\mathcal{O}((q+k)^k |G|^2)$ , the lemma follows. ◀

**Algorithm for the Decision version of Theorem 8.** Let  $\mathbf{I}$  be the given input CSP instance and let  $G$  be its incidence graph. We begin by setting  $q = \alpha(k, 2(k+1))$ , choosing the boundary  $Z$  to be the empty set and then executing the algorithm of Lemma 27 to compute a  $t$ -boundaried graph  $(G', Z')$  where  $G'$  is a subgraph of  $G$ ,  $|V(G')| > q$  and  $t \leq 2(k+1)$  such that  $G'$  has no  $(8^q, k+1)$ -separation. Next, we invoke Lemma 23 on the corresponding CSP instance, say  $\mathbf{I}'$ , to compute in time  $\mathcal{O}(\mathfrak{M}(k)|G|^2)$  a  $t$ -boundaried CSP instance  $\mathbf{I}''$  such that  $\mathbf{I}' \sim_{t,k} \mathbf{I}''$ . We then set  $\mathbf{I} = \mathbf{I}'' \oplus (\psi(G - (V(G') \setminus Z')))$  and recursively check for the presence of a strong backdoor set of width at most  $k$  for this instance. Since we strictly reduce the size of the instance in each step, the depth of the recursion is bounded linearly in the size of the initial input, implying FPT running time.

**Proof of Theorem 1 and Corollary 2.** Using the self-reducibility of the problem and the algorithm for the decision variant of Theorem 8, one can compute a strong backdoor set of width at most  $k$  (if it exists). Following this, one can execute the algorithm of Lemma 6 to solve CSP and #CSP. ◀

## 7 Concluding Remarks

We have introduced the notion of backdoor treewidth for CSP and #CSP by combining the two classical approaches of placing structural restrictions and language restrictions, respectively, on the input. Thus the presented results represent a new “hybrid” approach for solving CSP and #CSP. Our main result, Theorem 1, is quite broad as it covers all tractable finite constraint languages combined with the graph invariant treewidth. This can be seen as the base case of a general framework which combines a specific graph invariant of the torso graph with a specific class of constraint languages. Therefore, we hope it will stimulate further research in this direction.

---

### References

- 1 Stefan Arnborg, Bruno Courcelle, Andrzej Proskurowski, and Detlef Seese. An algebraic theory of graph reduction. *J. ACM*, 40(5):1134–1164, 1993. doi:10.1145/174147.169807.
- 2 Christian Bessiere, Clément Carbonnel, Emmanuel Hebrard, George Katsirelos, and Toby Walsh. Detecting and exploiting subproblem tractability. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. IJCAI/AAAI, 2013.
- 3 Hans L. Bodlaender and Babette de Fluiter. Reduction algorithms for constructing solutions in graphs with small treewidth. In Jin-Yi Cai and Chak Kuen Wong, editors, *COCOON’96*, LNCS, pages 199–208. Springer, 1996.
- 4 Hans L. Bodlaender and Torben Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM J. Comput.*, 27:1725–1746, 1998.
- 5 Hans L. Bodlaender and Babette van Antwerpen-de Fluiter. Reduction algorithms for graphs of small treewidth. *Inf. Comput.*, 167:86–119, 2001.
- 6 Andrei A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *J. of the ACM*, 53(1):66–120, 2006.
- 7 Andrei A. Bulatov. Complexity of conservative constraint satisfaction problems. *ACM Trans. Comput. Log.*, 12(4):Art. 24, 66, 2011.
- 8 Andrei A. Bulatov. The complexity of the counting constraint satisfaction problem. *J. of the ACM*, 60(5):Art 34, 41, 2013.
- 9 Andrei A. Bulatov, Andrei A. Krokhin, and Peter Jeavons. The complexity of maximal constraint languages. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 667–674. ACM, 2001.
- 10 Clément Carbonnel and Martin C. Cooper. Tractability in constraint satisfaction problems: a survey. *Constraints*, 21(2):115–144, 2016.
- 11 Clément Carbonnel, Martin C. Cooper, and Emmanuel Hebrard. On backdoors to tractable constraint languages. In *Principles and Practice of Constraint Programming – 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 224–239. Springer Verlag, 2014.
- 12 Rajesh Hemant Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. Designing FPT algorithms for cut problems using randomized con-

- tractions. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 460–469, 2012.
- 13 David Cohen, Peter Jeavons, and Marc Gyssens. A unified theory of structural tractability for constraint satisfaction problems. *J. of Computer and System Sciences*, 74(5):721–743, 2008.
  - 14 Martin C. Cooper, David A. Cohen, and Peter G. Jeavons. Characterising tractable constraints. *Artificial Intelligence*, 65(2):347–361, 1994.
  - 15 Bruno Courcelle. The monadic second-order logic of graphs. I. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.
  - 16 Y. Crama, O. Ekin, and P. L. Hammer. Variable and term removal from Boolean formulae. *Discr. Appl. Math.*, 75(3):217–230, 1997.
  - 17 Víctor Dalmau. A new tractable class of constraint satisfaction problems. In *AMAI, 6th International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, Florida, USA, January 5-7, 2000*, 2000.
  - 18 Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In Pascal Van Hentenryck, editor, *Principles and Practice of Constraint Programming – CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*, volume 2470 of *Lecture Notes in Computer Science*, pages 310–326. Springer Verlag, 2002.
  - 19 Babette de Fluiter. *Algorithms for Graphs of Small Treewidth*. PhD thesis, Utrecht University, 1997.
  - 20 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
  - 21 Tommy Färnqvist. Constraint optimization problems and bounded tree-width revisited. In Nicolas Beldiceanu, Narendra Jussien, and Eric Pinson, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems – 9th International Conference, CPAIOR 2012, Nantes, France, May 28 June 1, 2012. Proceedings*, volume 7298 of *Lecture Notes in Computer Science*, pages 163–179. Springer Verlag, 2012.
  - 22 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.
  - 23 Michael R. Fellows and Michael A. Langston. An analogue of the Myhill-Nerode theorem and its use in computing finite-basis characterizations (extended abstract). In *FOCS*, pages 520–525, 1989.
  - 24 Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, M. S. Ramanujan, and Saket Saurabh. Solving  $d$ -SAT via backdoors to small treewidth. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 630–641. SIAM, 2015.
  - 25 Eugene C. Freuder. A sufficient condition for backtrack-bounded search. *J. ACM*, 32(4):755–761, 1985.
  - 26 Robert Ganian, M. S. Ramanujan, and Stefan Szeider. Discovering archipelagos of tractability for constraint satisfaction and counting. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1670–1681, 2016.
  - 27 Serge Gaspers, Neeldhara Misra, Sebastian Ordyniak, Stefan Szeider, and Stanislav Zivny. Backdoors into heterogeneous classes of SAT and CSP. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27-31, 2014, Québec City, Québec, Canada.*, pages 2652–2658. AAAI Press, 2014.

- 28 Serge Gaspers, Sebastian Ordyniak, M. S. Ramanujan, Saket Saurabh, and Stefan Szeider. Backdoors to q-horn. *Algorithmica*, 74(1):540–557, 2016. doi:10.1007/s00453-014-9958-5.
- 29 Serge Gaspers and Stefan Szeider. Strong backdoors to bounded treewidth SAT. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 489–498. IEEE Computer Society, 2013.
- 30 G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *J. of Computer and System Sciences*, 64(3):579–627, 2002.
- 31 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. of the ACM*, 54(1), 2007.
- 32 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 479–488, 2011.
- 33 Pavol Hell and Jaroslav Nešetřil. On the complexity of  $H$ -coloring. *J. Comb. Theory, Ser. B*, 48(1):92–110, 1990.
- 34 Pavol Hell and Jaroslav Nešetřil. Colouring, constraint satisfaction, and complexity. *Computer Science Review*, 2(3):143–163, 2008.
- 35 Lane A. Hemaspaandra and Ryan Williams. SIGACT news complexity theory column 76: an atypical survey of typical-case heuristic algorithms. *SIGACT News*, pages 70–89, 2012.
- 36 Ken-ichi Kawarabayashi and Mikkel Thorup. The minimum  $k$ -way cut of bounded size is fixed-parameter tractable. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 160–169, 2011.
- 37 Phokion G. Kolaitis. Constraint satisfaction, databases, and logic. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 1587–1595. Morgan Kaufmann, 2003.
- 38 Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. of Computer and System Sciences*, 61(2):302–332, 2000. Special issue on the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (Seattle, WA, 1998).
- 39 Jens Lagergren. Upper bounds on the size of obstructions and intertwines. *J. Comb. Theory, Ser. B*, 73(1):7–40, 1998.
- 40 Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *J. of the ACM*, 60(6):Art. 42, 51, 2013.
- 41 Ugo Montanari. Networks of constraints: fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.
- 42 Naomi Nishimura, Prabhakar Ragde, and Stefan Szeider. Detecting backdoor sets with respect to Horn and binary clauses. In *Proceedings of SAT 2004 (Seventh International Conference on Theory and Applications of Satisfiability Testing, 10–13 May, 2004, Vancouver, BC, Canada)*, pages 96–103, 2004.
- 43 Igor Razgon and Barry O’Sullivan. Almost 2-SAT is fixed parameter tractable. *J. of Computer and System Sciences*, 75(8):435–450, 2009.
- 44 Marko Samer and Stefan Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010.
- 45 Marko Samer and Stefan Szeider. Constraint satisfaction with bounded treewidth revisited. *J. of Computer and System Sciences*, 76(2):103–114, 2010.
- 46 Thomas J. Schaefer. The complexity of satisfiability problems. In *Conference Record of the Tenth Annual ACM Symposium on Theory of Computing (San Diego, Calif., 1978)*, pages 216–226. ACM, 1978.

- 47 Ryan Williams, Carla Gomes, and Bart Selman. Backdoors to typical case complexity. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, IJCAI 2003*, pages 1173–1178. Morgan Kaufmann, 2003.
- 48 Ryan Williams, Carla Gomes, and Bart Selman. On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search. In *Informal Proc. of the Sixth International Conference on Theory and Applications of Satisfiability Testing, S. Margherita Ligure – Portofino, Italy, May 5-8, 2003 (SAT 2003)*, pages 222–230, 2003.





# On the Complexity of Partial Derivatives\*

Ignacio Garcia-Marco<sup>1</sup>, Pascal Koiran<sup>2</sup>, Timothée Pecatte<sup>3</sup>, and  
Stéphan Thomassé<sup>4</sup>

1 LIF and CNRS, Aix-Marseille Université, Marseille, France

iggarcia@ull.es

2 LIP,<sup>†</sup> Ecole Normale Supérieure de Lyon, Université de Lyon, Lyon, France

Pascal.Koiran@ens-lyon.fr

3 LIP, Ecole Normale Supérieure de Lyon, Université de Lyon, Lyon, France

Timothee.Pecatte@ens-lyon.fr

4 LIP, Ecole Normale Supérieure de Lyon, Université de Lyon, Lyon, France

Stephan.Thomasse@ens-lyon.fr

---

## Abstract

The method of partial derivatives is one of the most successful lower bound methods for arithmetic circuits. It uses as a complexity measure the dimension of the span of the partial derivatives of a polynomial. In this paper, we consider this complexity measure as a computational problem: for an input polynomial given as the sum of its nonzero monomials, what is the complexity of computing the dimension of its space of partial derivatives?

We show that this problem is  $\sharp\text{P}$ -hard and we ask whether it belongs to  $\sharp\text{P}$ . We analyze the “trace method”, recently used in combinatorics and in algebraic complexity to lower bound the rank of certain matrices. We show that this method provides a polynomial-time computable lower bound on the dimension of the span of partial derivatives, and from this method we derive closed-form lower bounds. We leave as an open problem the existence of an approximation algorithm with reasonable performance guarantees.

**1998 ACM Subject Classification** F.1.3 Complexity Measures and Classes, F.2 Analysis of algorithms and problem complexity

**Keywords and phrases** counting complexity, simplicial complex, lower bounds, arithmetic circuits

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.37

## 1 Introduction

Circuit lower bounds against a class of circuits  $\mathcal{C}$  are often obtained by defining an appropriate complexity measure which is small for small circuits of  $\mathcal{C}$  but is high for some explicit “hard function.” For arithmetic circuits, one of the most successful complexity measures is based on partial derivatives. Sums of powers of linear forms provide the simplest model where the method of partial derivatives can be presented (see for instance Chapter 10 of the survey by Chen, Kayal and Wigderson [2]). In this model, a homogeneous polynomial  $f(x_1, \dots, x_n)$  of degree  $d$  is given by an expression of the form:

$$f(x_1, \dots, x_n) = \sum_{i=1}^r l_i(x_1, \dots, x_n)^d \quad (1)$$

---

\* Work supported by ANR project CompA (code ANR-13-BS02-0001-01).

<sup>†</sup> UMR 5668 ENS Lyon, CNRS, UCBL, INRIA.



© Ignacio Garcia-Marco, Pascal Koiran, Timothée Pecatte, and Stéphan Thomassé;  
licensed under Creative Commons License CC-BY

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 37; pp. 37:1–37:13

Leibniz International Proceedings in Informatics



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

where the  $l_i$ 's are linear functions. The smallest possible  $r$  is often called the Waring rank of  $f$  in the algebra literature. One takes as complexity measure  $\dim \partial^{=k} f$ , where  $\partial^{=k} f$  denotes the linear space of polynomials spanned by the partial derivatives of  $f$  of order  $k$ . For any  $k \leq d$ , the derivatives of order  $k$  of a  $d$ -th power of a linear form  $l(x_1, \dots, x_n)$  are constant multiples of  $l^{d-k}$ . Therefore, by linearity of derivatives we have for any  $k \leq d$  the lower bound  $r \geq \dim \partial^{=k} f$  on the Waring rank of  $f$ .

The method of partial derivatives was introduced in the complexity theory literature by Nisan and Wigderson [14], where lower bounds were given for more powerful models than (1) such as e.g. depth 3 arithmetic circuits. In such a circuit, the  $d$ -th powers in (1) are replaced by products of  $d$  affine functions. We then have [14] the lower bound  $r \geq (\dim \partial^* f)/2^d$ , where  $r$  denotes as in (1) the fan-in of the circuit's output gate and  $\partial^* f$  denotes the space spanned by partial derivatives of all order. More recently, a number of new lower bound results were obtained using a refinement of the method of partial derivatives. These new results are based on "shifted partial derivatives" (see the continuously updated online survey maintained by Saptharishi [17] for an extensive list of references), but we will stick to "unshifted" derivatives in this paper.

Partial derivatives can also be used for *upper bound* results: see in particular Theorem 5 in [9] for an algorithm that constructs a representation in the Waring model (1) of a polynomial given by a black box. To learn more on the complexity of circuit reconstruction for various classes of arithmetic circuits one may consult Chapter 5 of the survey by Shpilka and Yehudayoff [18].

### Our contributions

In this paper we consider the dimension of the set of partial derivatives as a computational problem and provide the first results (that we are aware of) on its complexity. This is quite a natural problem since, as explained above, the knowledge of this dimension for an input polynomial  $f$  provides estimates on the circuit size of  $f$  for several classes of arithmetic circuits. We assume that the input polynomial  $f$  is given in the sparse representation (also called "expanded representation"), i.e., as the sum of its nonzero monomials. We show in Section 4 that computing  $\dim \partial^* f$  is hard for Valiant's [20] counting class  $\#\text{P}$ . This remains true even if  $f$  is multilinear, homogeneous and has only 0/1 coefficients. The precise complexity of this problem remains open, in particular we do not know whether computing  $\dim \partial^* f$  is in  $\#\text{P}$ .

As an intermediate step toward our  $\#\text{P}$ -hardness result, we obtain a result of independent interest for a problem of topological origin: computing the number of faces in an abstract simplicial complex. Our  $\#\text{P}$ -hardness proof for this problem proceeds by reduction from counting the number of independent sets in a graph, a well-known  $\#\text{P}$ -complete problem [15]. It is inspired by the recent proof [16] that computing the Euler characteristic of abstract simplicial complexes is  $\#\text{P}$ -complete.

Since the  $\#\text{P}$ -hardness result rules out an efficient algorithm for the exact computation of  $\dim \partial^* f$ , it is of interest to obtain efficiently computable upper and lower bounds for this quantity and for  $\dim \partial^{=k} f$ . Upper bounds are easily obtained from the linearity of derivatives. In Section 2 we give a lower bound that is based on the consideration of a single "extremal" monomial of  $f$ . In particular, for a multilinear homogeneous polynomial of degree  $d$  with  $s$  monomials we have  $\binom{d}{k} \leq \dim \partial^{=k} f \leq s \binom{d}{k}$  for every  $k$ . In Section 3 we provide lower bounds that take all monomials of  $f$  into account. Depending on the choice of the input polynomial, these lower bounds may be better or worse than the lower bound of Section 2. The lower bounds of Section 3 are based on the "trace method." This method was recently used in [10, 11] to lower bound the dimension of *shifted* partial derivatives of a specific "hard"

polynomial, the so-called Nisan-Wigderson polynomial. In [10] this method is attributed to Noga Alon [1].

In a nutshell, the principle of the trace method is as follows. Suppose that we want to lower bound the rank of a matrix  $M$ . In this paper,  $M$  will be the matrix of partial derivatives of a polynomial  $f(x_1, \dots, x_n)$ . From  $M$ , we construct the symmetric matrix  $B = M^T \cdot M$ . We have  $\text{rank}(M) \geq \text{rank}(B)$ , with equality if the ranks are computed over the field of real numbers. In the trace method, we replace  $\text{rank}(M) = \text{rank}(B)$  by the “proxy rank”  $\text{Tr}(B)^2/\text{Tr}(B^2)$ . This is legitimate due to the inequality

$$\text{rank}(B) \geq \text{Tr}(B)^2/\text{Tr}(B^2), \quad (2)$$

which follows from the Cauchy-Schwarz inequality applied to the eigenvalues of  $B$ . It is often easier to lower bound the proxy rank than to lower bound the rank directly. In Section 3 we will see that the proxy rank can be computed in polynomial time. This is not self-evident because  $B$  may be of size exponential in the number  $n$  of variables of  $f$ . By contrast, as explained above computing  $\text{rank}(B)$  over the field of real numbers is  $\sharp\text{P}$ -hard.

### Organization of the paper

In Section 2 we set up the notation for the rest of the paper, and give some elementary estimates. In particular, Theorem 1 provides a lower bound that relies on the consideration of a single extremal monomial of  $f$ . Section 3 is devoted to the trace method. We use this method to derive closed-form lower bounds on the dimension of the space of partial derivatives, and compare them to the lower bound from Theorem 1. In Section 3.2 we show that the “proxy rank”  $\text{Tr}(B)^2/\text{Tr}(B^2)$  is computable in polynomial time. In Section 3.3 we show that the trace method behaves very poorly on elementary symmetric polynomials: for certain settings of parameters, the matrix of partial derivatives has full rank but the trace method can only show that its rank is larger than 1. Finally, we show in Section 4 that it is  $\sharp\text{P}$ -hard to compute  $\dim \partial^* f$  and to compute the number of faces in an abstract simplicial complex.

### Open problems

Here are three of the main problems that are left open by this work.

1. Give a nontrivial upper bound on the complexity of computing  $\dim \partial^* f$  and  $\dim \partial^{=k} f$ . In particular, are these two problems in  $\sharp\text{P}$ ?
2. Give an efficient algorithm that approximates  $\dim \partial^* f$  or  $\dim \partial^{=k} f$ , and comes with a reasonable performance guarantee (or show that such an algorithm does not exist). The proxy rank  $\text{Tr}(B)^2/\text{Tr}(B^2)$  is efficiently computable, but certainly does not fit the bill due to its poor performance on symmetric polynomials. For counting the number of independent sets in a graph (the starting point of our reductions), there is already a significant amount of work on approximation algorithms [13, 4] and hardness of approximation [13, 3].
3. We recalled at the beginning of the introduction that partial derivatives are useful as a complexity measure to prove lower bounds against several classes of arithmetic circuits. We saw that computing this measure is hard, but is it hard to compute the Waring rank of a homogeneous polynomial  $f$  given in expanded form, or to compute the size of the smallest (homogeneous) depth 3 circuit for  $f$ ? The former problem is conjectured to be NP-hard already for polynomials of degree 3: see Conjecture 13.2 in [8] which is formulated in the language of symmetric tensors.

**2 Elementary bounds**

We use the notation  $\partial_\beta f$  for partial derivatives of a polynomial  $f(x_1, \dots, x_n)$ . Here  $\beta$  is a  $n$ -tuple of integers, and  $\beta_i$  is the number of times that we differentiate  $f$  with respect to  $x_i$ . We denote by  $\partial^{=k} f$  the linear space spanned by the partial derivatives of  $f$  of order  $k$ , and by  $\partial^* f$  the space spanned by partial derivatives of all order. For  $\alpha \in \{0, 1\}^n$ , we denote by  $x^\alpha$  the multilinear monomial  $x_1^{\alpha_1} \cdots x_n^{\alpha_n}$ . More generally, if  $\alpha$  is a  $n$ -tuple of integers,  $x^\alpha$  denotes the monomial  $x_1^{\alpha_1} \cdots x_n^{\alpha_n} / (\alpha_1! \cdots \alpha_n!)$ . These monomials form a basis of the space  $\mathbb{R}[x_1, \dots, x_n]$  of real polynomials in  $n$  variables, which we refer to as the “scaled monomial basis.” Dividing by the constant  $\alpha_1! \cdots \alpha_n!$  is convenient since differentiation takes the simple form:  $\partial_\beta x^\alpha = x^{\alpha-\beta}$ . We agree that  $x^{\alpha-\beta} = 0$  if one of the components of  $\alpha - \beta$  is negative.

For a monomial  $f = x^\alpha$ ,  $\dim \partial^* x^\alpha = \prod_{i=1}^n (\alpha_i + 1)$ . One can compute  $\dim \partial^{=k} x^\alpha$  by dynamic programming thanks to the recurrence relation:

$$\dim \partial^{=k} x^\alpha = \sum_{j=0}^{\alpha_1} \dim \partial^{=k-j} (x_2^{\alpha_2} \cdots x_n^{\alpha_n}).$$

It takes altogether  $O((\deg f)^2)$  additions to compute the  $\deg(f) + 1$  numbers  $\dim \partial^{=k} f$  for  $k = 0, \dots, \deg(f)$ . Equivalently,  $\dim \partial^{=k} x^\alpha$  can be computed as the coefficient of  $t^k$  in the polynomial

$$(1 + t + \dots + t^{\alpha_1}) \cdot (1 + t + \dots + t^{\alpha_2}) \cdots (1 + t + \dots + t^{\alpha_n}).$$

For a polynomial with more than one monomial, one can obtain simple upper bounds thanks to the linearity of derivatives since  $\dim \partial^*(f+g) \leq \dim \partial^* f + \dim \partial^* g$  and  $\dim \partial^{=k}(f+g) \leq \dim \partial^{=k} f + \dim \partial^{=k} g$ . Lower bounding the dimension of the space of partial derivatives is slightly less immediate.

► **Theorem 1.** *For any polynomial  $f$  there is a monomial  $m$  in  $f$  such that  $\dim \partial^{=k} f \geq \dim \partial^{=k} m$  for every  $k$ . In particular, if all monomials in  $f$  contain at least  $r$  variables then  $\dim \partial^{=k} f \geq \binom{r}{k}$  for every  $k$ .*

**Proof.** The second claim clearly follows from the first claim. Let  $n$  be the number of variables in  $f$ . In order to find the monomial  $m$ , we fix a total order  $\leq$  on  $n$ -tuple of integers which is compatible with addition, for instance the lexicographic order (different orders may lead to different  $m$ 's). We will use  $\leq$  to order monomials as well as tuples  $\beta$  in partial derivatives such as  $\partial_\beta f$ . We will also use the partial order  $\subseteq$  defined by:  $\beta \subseteq \alpha$  iff  $\beta_i \leq \alpha_i$  for all  $i = 1, \dots, n$ . Let  $m = x^\alpha$  be the smallest monomial for  $\leq$  with a nonzero coefficient in  $f$ .

To complete the proof of the theorem, we just need to show that the partial derivatives  $\partial_\beta f$  where  $\beta \subseteq \alpha$  are linearly independent. The dimension of the space spanned by these partial derivatives is equal to the rank of a certain matrix  $M$ . The rows of  $M$  are indexed by the  $n$ -tuples  $\beta$  such that  $\beta \subseteq \alpha$ , and row  $\beta$  contains the coordinates of  $\partial_\beta f$  in the scaled monomial basis  $(x^\gamma)$ . If  $f = \sum_\gamma a_\gamma x^\gamma$ , we therefore have  $M_{\beta, \gamma-\beta} = a_\gamma$ . Let us order the rows and columns of  $M$  according to  $\leq$ . We have seen that  $M$  contains a nonzero coefficient in row  $\beta$  and column  $\alpha - \beta$ . This coefficient is strictly to the left of any nonzero coefficient in any row above  $\beta$ . Indeed, we have  $\alpha - \beta < \alpha' - \beta'$  if  $\alpha' \geq \alpha$  and  $\beta' < \beta$ . Our matrix is therefore in row echelon form, and does not contain any identically zero row. It is therefore of full row rank. ◀

► **Remark.** Recall that the Newton polytope of  $f$  is the convex hull of the  $n$ -tuples of exponents of monomials of  $f$ . By changing the order  $\leq$  in the proof of Theorem 1 we can take for  $m$  any vertex of the Newton polytope.

Theorem 1 lower bounds  $\dim \partial^{=k} f$  by the same dimension computed for a suitable monomial of  $f$ . This is of course tight if  $f$  has a single monomial. We note that adding more monomials does not necessarily increase  $\dim \partial^{=k} f$ . For instance, the polynomial  $f = \prod_{i=1}^d \sum_{j=1}^q x_{ij}$  has  $q^d$  monomials but  $\dim \partial^{=k} f$  remains equal to  $\binom{d}{k}$  for any  $q$ .

► **Corollary 2.** *For a multilinear homogeneous polynomial of degree  $d$  with  $s$  monomials we have  $\binom{d}{k} \leq \dim \partial^{=k} f \leq s \binom{d}{k}$  for every  $k$ .*

**Proof.** The upper bound follows from the linearity of derivatives, and the lower bound from Theorem 1. ◀

### 3 The trace method

The lower bound on  $\dim \partial^{=k} f$  in Theorem 1 takes a single monomial of  $f$  into account. In this section we give a more “global” result which takes all monomials into account. We will in fact lower bound the dimension of a subspace of  $\partial^{=k} f$ , spanned by partial derivatives of the form  $\partial_I f$  where  $I \in \{0, 1\}^n$ . In other words, we will differentiate at most once with respect to any variable.<sup>1</sup> We can of course view  $I$  as a subset of  $[n]$  rather than as a vector in  $\{0, 1\}^n$ .

We form a matrix  $M$  of partial derivatives as in the proof of Theorem 1. The rows of  $M$  are indexed by subsets of  $[n]$  of size  $k$ , and row  $I$  contains the expansion of  $\partial_I f$  in the basis  $(x^J)$ . If  $f = \sum_j a_j x^J$ , we have seen in Section 2 that  $M_{I,J} = a_{I+J}$ . In order to lower bound the rank of  $M$ , we will apply the following lemma to the symmetric matrix  $B = M^T M$ .

► **Lemma 3.** *For any real symmetric matrix  $B \neq 0$  we have*

$$\text{rank}(B) \geq \frac{(\text{Tr} B)^2}{\text{Tr}(B^2)}.$$

Lemma 3 is easily obtained by applying the Cauchy-Schwarz inequality to the vector of nonzero eigenvalues of  $B$ . Note that  $B = M^T M$  has same rank as  $M$  since we have:  $x^T B x = 0 \Leftrightarrow M x = 0$  for any vector  $x$ .

We first consider the case of polynomials with 0/1 coefficients, for which we have the following lower bound.

► **Theorem 4.** *For  $f$  a real polynomial with 0/1 coefficients we have*

$$\dim \partial^{=k} f \geq \frac{\sum_{P \in \mathcal{M}} \binom{\text{sup}(P)}{k}}{|\mathcal{M}|^2} \tag{3}$$

where  $\mathcal{M}$  denotes the set of monomials occurring in  $f$ , and  $\text{sup}(P)$  the number of distinct variables occurring in monomial  $P$ .

The right-hand side of (3) is sandwiched between  $\binom{\text{supmin}}{k}/|\mathcal{M}|$  and  $\binom{\text{supmax}}{k}/|\mathcal{M}|$ , where  $\text{supmin}$  and  $\text{supmax}$  denote respectively the minimum and maximum number of variables occurring in a monomial of  $f$ . Theorem 1 provides a better lower bound (by a factor of  $|\mathcal{M}|$ )

<sup>1</sup> One could lift this restriction and derive similar results for the “full” matrix of  $k$ -th order derivatives, i.e., for the case where several differentiations with respect to the same variable are allowed. This would have the effect of replacing the binomial coefficients  $\binom{\text{sup}(P)}{k}$  in the lower bounds of the present section by  $\dim \partial^{=k} P$ . Here  $P$  denotes a monomial of  $f$ ; we have explained at the beginning of Section 2 how to compute  $\dim \partial^{=k} P$ . We will stick here to a single differentiation for the sake of notational simplicity.

## 37:6 On the Complexity of Partial Derivatives

when all the monomials of  $f$  have supports of same size. Theorem 4 becomes interesting when all but a few monomials in  $f$  have large support. Indeed, the presence of a few monomials of small support can ruin the lower bound of Theorem 1.

► **Example 5.** Let  $f(x_1, \dots, x_n) = x_1.x_2.\dots.x_n + \sum_{i=1}^n x_i^n$ . The Newton polytope of  $f$  is an  $n$ -simplex whose vertices correspond to the monomials  $x_1^n, \dots, x_n^n$ . The point corresponding to the monomial  $x_1.x_2.\dots.x_n$  is the barycenter of this simplex, and in particular it is not a vertex of the Newton polytope. As a result, by Remark 2 the lower bound method of Theorem 1 can only show that  $\dim \partial^{=k} f \geq 1$ . Theorem 4 shows the better lower bound:

$$\dim \partial^{=k} f \geq \frac{\binom{n}{k} + n}{(n+1)^2}.$$

It is not hard to check by a direct calculation that for this example, the correct value of  $\dim \partial^{=k} f$  is:

- 1 for  $k \in \{0, n\}$ ;
- $n$  for  $k \in \{1, n-1\}$ ;
- $\binom{n}{k} + n$  for  $2 \leq k \leq n-2$ .

Let us now proceed with the proof of Theorem 4. In view of Lemma 3, we need a lower bound on  $\text{Tr}(B)$  and an upper bound on  $\text{Tr}(B^2)$ .

► **Lemma 6.**  $\text{Tr}(B) = \sum_{P \in \mathcal{M}} \binom{\text{sup}(P)}{k}$ .

**Proof.** By definition of  $B$ ,  $\text{Tr}(B) = \sum_J (M^T.M)_{J,J} = \sum_{I,J} M_{I,J}^2$ . Since  $M_{I,J} \in \{0, 1\}$ , this is nothing but the number of nonzero entries in  $M$ . Monomial  $P$  contributes  $\binom{\text{sup}(P)}{k}$  such entries and they are all distinct. ◀

► **Lemma 7.**  $\text{Tr}(B^2) \leq |\mathcal{M}|^2 \sum_{P \in \mathcal{M}} \binom{\text{sup}(P)}{k}$ .

**Proof.** Since  $B$  is symmetric,  $\text{Tr}(B^2) = \sum_{K,L} (B_{K,L})^2$ . By definition of  $B$ ,  $B_{K,L} = \sum_I M_{I,K}.M_{I,L}$ . Therefore

$$\text{Tr}(B^2) = \sum_{I,J,K,L} M_{I,K}.M_{I,L}.M_{J,K}.M_{J,L}.$$

In this formula,  $I, J$  range over row indices (subsets of  $[n]$  of size  $k$ ), and  $K, L$  range over column indices. Hence  $\text{Tr}(B^2)$  is equal to the number of quadruples  $(I, J, K, L)$  such that all 4 entries  $M_{I,K}, M_{I,L}, M_{J,K}, M_{J,L}$  are nonzero. Let us say that a quadruple is *valid* if this condition is satisfied. A quadruple is valid if and only if the 4 coefficients  $a_{I+K}, a_{I+L}, a_{J+K}, a_{J+L}$  are nonzero. This implies that there are at most  $|\mathcal{M}|^2 \sum_{P \in \mathcal{M}} \binom{\text{sup}(P)}{k}$  valid quadruples. Let us indeed denote by  $P, Q, R$  the 3  $n$ -tuples  $I+K, I+L, J+K$ . For every fixed  $P$  we have at most  $\binom{\text{sup}(P)}{k}$  choices for  $I$  since  $I$  is contained in the support of  $P$ , and at most  $|\mathcal{M}|^2$  choices for the pair  $(Q, R)$ . The result follows since the quadruple  $(I, J, K, L)$  is completely determined by the choices of  $P, Q, R$  and  $I$ : we must have  $K = P - I, L = Q - I, J = R - K$ . ◀

Theorem 4 follows immediately from Lemmas 3 to 7.

### 3.1 Extension to real coefficients

In this section we generalize Theorem 4 to polynomials with real coefficients. Theorem 8 could itself be generalized to polynomials with complex coefficients by working with a Hermitian matrix in Lemma 3 rather than with a symmetric matrix.

► **Theorem 8.** *For any real polynomial  $f$  we have*

$$\dim \partial^k f \geq \frac{\sum_{P \in \mathcal{M}} \binom{\text{sup}(P)}{k} a_P^2}{|\mathcal{M}| \sum_{P \in \mathcal{M}} a_P^2} \tag{4}$$

where  $\mathcal{M}$  denotes the set of monomials occurring in  $f$ .

To make sense of the lower bound in this theorem, it is helpful to look at a couple of special cases. If the coefficients  $a_P$  all have the same absolute value, e.g.,  $|a_P| = 1$  for all  $P \in \mathcal{M}$ , the right-hand side of (4) reduces to  $\sum_{P \in \mathcal{M}} \binom{\text{sup}(P)}{k} / |\mathcal{M}|^2$ . This is exactly the lower bound in Theorem 4 (but now the coefficients of  $f$  may be in  $\{-1, 0, 1\}$  rather than  $\{0, 1\}$ ).

Our lower bound becomes weaker when the vectors  $(\binom{\text{sup}(P)}{k})_{P \in \mathcal{M}}$  and  $(a_P^2)_{P \in \mathcal{M}}$  are approximately orthogonal. This can happen when the monomials with large support have small coefficients. In this case, as should be expected, Lemma 3 is effectively unable to detect the presence of monomials of large support. A probabilistic analysis shows that this bad behavior is atypical. Consider for instance the following semirandom model: we first choose a set  $\mathcal{M}$  of monomials in some arbitrary (worst case) way, and then the  $a_P$  are drawn independently at random from some common probability distribution such that  $\Pr[a_P = 0] = 0$ .

► **Corollary 9.** *Let  $L(f) = \frac{\sum_{P \in \mathcal{M}} \binom{\text{sup}(P)}{k} a_P^2}{|\mathcal{M}| \sum_{P \in \mathcal{M}} a_P^2}$  be the lower bound on the right-hand side of (4).*

*In the semirandom model described above, the expectation of  $L(f)$  is:*

$$E[L(f)] = \sum_{P \in \mathcal{M}} \binom{\text{sup}(P)}{k} / |\mathcal{M}|^2.$$

We omit the (simple) proof due to lack of space. Note that we obtain for  $E[L(f)]$  the lower bound from the case where  $|a_P| = 1$  for all  $P \in \mathcal{M}$ .

The remainder of this section is devoted to the proof of Theorem 8. We follow the proof of Theorem 4. In particular, we still differentiate at most once with respect to each variable, we define the same matrix  $M$  of partial derivatives and the symmetric matrix  $B = M^T \cdot M$ . We again have  $\dim \partial^k f \geq \text{rank}(M) = \text{rank}(B)$ ; hence Theorem 8 follows from Lemma 3 and from the next two lemmas.

► **Lemma 10.**  $\text{Tr}(B) = \sum_{P \in \mathcal{M}} \binom{\text{sup}(P)}{k} a_P^2$ .

**Proof.** By the proof of Lemma 6,  $\text{Tr}(B)$  is equal to the sum of squared entries of  $M$ ; and we have  $M_{I, P-I} = a_P$  for each set  $I$  of size  $k$  contained in the support of  $P$ . ◀

► **Lemma 11.**  $\text{Tr}(B^2) \leq |\mathcal{M}| \text{Tr}(B) (\sum_{R \in \mathcal{M}} a_R^2)$ .

**Proof.** By the proof of Lemma 7,

$$\text{Tr}(B^2) = \sum_{(I, J, K, L)} a_{I+K} \cdot a_{I+L} \cdot a_{J+K} \cdot a_{J+L}. \tag{5}$$

### 37:8 On the Complexity of Partial Derivatives

Here  $I, J$  range over row indices of  $M$  while  $K, L$  range over column indices. As in the proof of Lemma 7, we call such a quadruple “valid” if the coefficients  $a_{I+K}, a_{I+L}, a_{J+K}, a_{J+L}$  are all nonzero. Let  $\mathcal{V}$  be the set of valid quadruples. Trying to mimic the proof of Lemma 7, we will write

$$\mathrm{Tr}(B^2) = \sum_{(P,Q,R,I) \in \mathcal{U}} a_P \cdot a_Q \cdot a_R \cdot a_{Q+R-P} \quad (6)$$

where  $\mathcal{U}$  is the set of quadruples  $(P, Q, R, I)$  such that:

1.  $P, Q, R$  and  $Q + R - P$  belong to  $\mathcal{M}$  (the set of monomials of  $f$ ) and  $I$  is a row index of  $M$ , i.e.,  $I \in \{0, 1\}^n$  and  $|I| = k$ .
2.  $I \leq P$  and  $I \leq Q$ .
3. There exists a (unique) row index  $J$  such that  $P - I = R - J$ .

Equation (6) follows from (5) due to the following one-to-one correspondence between quadruples of  $\mathcal{U}$  and  $\mathcal{V}$ :

- (i) Given a quadruple  $(I, J, K, L) \in \mathcal{V}$ , set  $P = I + K$ ,  $Q = I + L$ ,  $R = J + K$ . The quadruple  $(P, Q, R, I)$  is in  $\mathcal{U}$  since  $Q + R - P = J + L$  and  $P - I = R - J = K$ .
- (ii) A quadruple  $(P, Q, R, I) \in \mathcal{U}$  has a unique preimage  $(I, J, K, L) \in \mathcal{V}$ , which is obtained as follows. A preimage must satisfy  $K = P - I$ ,  $L = Q - I$ ,  $J = R - K$ . This defines a quadruple  $(I, J, K, L)$  such that  $P - I = R - J$ , so  $J$  must be a row index by condition 3 in the definition of  $\mathcal{U}$ :  $J \in \{0, 1\}^n$  and  $|J| = k$ . It follows that  $(I, J, K, L) \in \mathcal{V}$  and that this quadruple is indeed a preimage of  $(P, Q, R, I)$ .

Since  $2a_P \cdot a_Q \cdot a_R \cdot a_{Q+R-P} \leq (a_P \cdot a_R)^2 + (a_Q \cdot a_{Q+R-P})^2$ , it follows from (6) that

$$2\mathrm{Tr}(B^2) \leq \sum_{(P,Q,R,I) \in \mathcal{U}} a_P^2 \cdot a_R^2 + \sum_{(P,Q,R,I) \in \mathcal{U}} a_Q^2 \cdot a_{Q+R-P}^2. \quad (7)$$

The first sum is upper bounded by

$$|\mathcal{M}| \cdot \left( \sum_{P \in \mathcal{M}} \binom{\sup(P)}{k} a_P^2 \right) \cdot \left( \sum_{R \in \mathcal{M}} a_R^2 \right) \quad (8)$$

since there are at most  $|\mathcal{M}|$  choices for  $Q$  and we must have  $I \leq P$  for a quadruple in  $\mathcal{U}$ . This is equal to

$$|\mathcal{M}| \mathrm{Tr}(B) \left( \sum_{R \in \mathcal{M}} a_R^2 \right)$$

by Lemma 10. Likewise, the second sum in (7) is upper bounded by

$$\sum_{P,Q,R \in \mathcal{M}} \binom{\sup(Q)}{k} a_Q^2 a_{Q+R-P}^2$$

since we have  $I \leq Q$  for a quadruple in  $\mathcal{U}$ . For any fixed  $Q \in \mathcal{M}$ ,  $\sum_{P,R \in \mathcal{M}} a_{Q+R-P}^2 \leq |\mathcal{M}| \cdot \sum_{S \in \mathcal{M}} a_S^2$  since each term  $a_S^2$  on the right-hand side can appear at most  $|\mathcal{M}|$  times on the left-hand side. We conclude that the second sum in (7) admits the same upper bound (8) as the first sum, and the lemma is proved. ◀

### 3.2 Polynomial-time computable lower bounds

The lower bound

$$L(f) = \frac{\sum_{P \in \mathcal{M}} \binom{\sup(P)}{k} a_P^2}{|\mathcal{M}| \sum_{P \in \mathcal{M}} a_P^2}$$



in Theorem 8 is clearly computable in polynomial time from  $f$  and  $k$ . Recall that we have obtained this lower bound by constructing a symmetric matrix  $B$  such that  $\dim \partial^{=k} f \geq \text{Tr}(B)^2 / \text{Tr}(B^2) \geq L(f)$ . The quantity  $\text{Tr}(B)^2 / \text{Tr}(B^2)$  is therefore a better lower bound on  $\dim \partial^{=k} f$  than  $L(f)$ . Like  $L(f)$ , it turns out to be computable in polynomial time. This is not self-evident because  $B$  may be of exponential size (which is the source of the  $\sharp\text{P}$ -hardness result in the next section).

► **Theorem 12.** *There is an algorithm which, given  $f$  and  $k$ , computes the lower bound  $\text{Tr}(B)^2 / \text{Tr}(B^2)$  on  $\dim \partial^{=k} f$  in polynomial time (as in the rest of this paper, we assume that  $f$  is given as the sum of its nonzero monomials).*

**Proof.** We build on the proof of Theorem 8. Lemma 10 shows that  $\text{Tr}(B)$  can be computed in polynomial time, so it remains to do the same for  $\text{Tr}(B^2)$ . In the proof of Lemma 11, we have defined a set of quadruples  $\mathcal{U}$  such that

$$\text{Tr}(B^2) = \sum_{(P,Q,R,I) \in \mathcal{U}} a_P \cdot a_Q \cdot a_R \cdot a_{Q+R-P}.$$

This can be rewritten as:

$$\text{Tr}(B^2) = \sum_{(P,Q,R) \in \mathcal{M}} N(P, Q, R) \cdot a_P \cdot a_Q \cdot a_R \cdot a_{Q+R-P}$$

where we denote by  $N(P, Q, R)$  the number of row indices  $I$  such that  $(P, Q, R, I) \in \mathcal{U}$ . It therefore remains to show that  $N(P, Q, R)$  can be computed in polynomial time. Toward this goal we make two observations.

- (i) Condition 2 in the definition of  $\mathcal{U}$  means that  $I \leq \min(P, Q)$ , where the  $n$ -tuple  $\min(P, Q)$  is the coordinatewise minimum of  $P$  and  $Q$ .
- (ii) The equality  $P - I = R - J$  in condition 3 is equivalent to  $P - R = I - J$ , hence  $P - R \in \{-1, 0, 1\}^n$  since  $I, J \in \{0, 1\}^n$ . Moreover, since  $I$  and  $J$  each have  $k$  nonzero coordinates,  $P - R$  must contain the same number of 1's and  $-1$ 's. By observation (i), the positions of 1's must be positive in  $\min(P, Q)$ .

We can therefore compute  $N(P, Q, R)$  as follows.

1. If  $Q + R - P$  is not a monomial of  $f$ ,  $N(P, Q, R) = 0$ .
2. If  $P - R$  is not in  $\{-1, 0, 1\}^n$ ,  $N(P, Q, R) = 0$ .
3. If  $P - R$  does not contain the same number of 1's and  $-1$ 's,  $N(P, Q, R) = 0$ .
4. If some of the positions of 1's in  $P - R$  contain a 0 in  $\min(P, Q)$ ,  $N(P, Q, R) = 0$ .
5. Let  $\text{ones}(P, R)$  be the number of 1's in  $P - R$  and  $\text{zeros}(P, Q, R)$  the number of 0's in  $P - R$  such that we have a positive entry in  $\min(P, Q)$  at the same position. Then  $N(P, Q, R) = \binom{\text{zeros}(P, Q, R)}{k - \text{ones}(P, R)}$ : from the relation  $P - R = I - J$ , the positions with a 1 in  $P - R$  must contain a 1 in  $I$ . So it remains to choose the remaining  $k - \text{ones}(P, R)$  nonzero positions of  $I$ . We can only choose them among the positions that are positive in  $\min(P, Q)$  (by (i)) and contain a 0 in  $P - R$ . ◀

### 3.3 Elementary symmetric polynomials

A natural question is whether the inequality  $\text{rank}(B) \geq \frac{(\text{Tr} B)^2}{\text{Tr}(B^2)}$  in Lemma 3 is tight when  $B = M^T \cdot M$  and  $M$  comes from a partial derivatives matrix. It is well known that this inequality is in general far from tight, since it is obtained by means of the Cauchy-Schwarz inequality. However in our case, due to the particular shape of the matrix  $B$ , it is not a priori clear whether a large gap can exist between  $\text{rank}(B)$  and  $\frac{(\text{Tr} B)^2}{\text{Tr}(B^2)}$ . In the following, we

## 37:10 On the Complexity of Partial Derivatives

show that arbitrarily large gaps can indeed be achieved. Our source of examples are the elementary symmetric polynomials  $Sym_{d,n}(x_1, \dots, x_n) = \sum_{|I|=d} x^I$ .

Here the vector  $I$  of exponents belongs to  $\{0, 1\}^n$ , and  $x^I$  denotes as usual the multilinear monomial  $x_1^{i_1} \cdots x_n^{i_n}$ .

More precisely, we will show the following.

► **Proposition 13.** *For any fixed positive integers  $d, k < d$ , the family of polynomials  $f_n = Sym_{d,n}$  has the following property: if we consider  $u_n = \text{rank}(B_n) = \dim \partial^{=k} f_n$  the sequence of dimensions of partial derivatives, and  $v_n = \frac{(\text{Tr} B_n)^2}{\text{Tr}(B_n^2)}$  the sequence of lower bounds for the dimension, we have that  $v_n \rightarrow 1$ , whereas  $B_n$  is of full rank and, hence,  $u_n \rightarrow +\infty$ .*

Note that since  $d$  is fixed, the polynomial  $f_n$  is sparse: it contains only  $n^{O(1)}$  monomials.

**Proof.** The matrix  $M$  of partial derivatives of  $f_n$  has only 0/1-coefficients and the coefficient  $M_{I,J}$  is non-zero iff  $I \cap J = \emptyset$  (with  $|I| = k, |J| = d - k$ ). This matrix is commonly known as the *disjointness matrix* and has proved useful in communication complexity [12] and of course in algebraic complexity [14] for the study of elementary symmetric polynomials.<sup>2</sup> In particular, by [6], we have that  $M$  is of full rank, i.e., that  $u_n = \dim \partial^{=k} f_n = \min\{\binom{n}{k}, \binom{n}{d-k}\}$ . This directly implies that  $u_n \rightarrow +\infty$ .

We already know that  $v_n \geq 1$  for all  $n$ , so we only need to compute an upper bound on  $v_n$  that tends to 1 to obtain  $v_n \rightarrow 1$ . To do so, we first compute the coefficients of the matrix  $B = M^T.M$ :

$$B_{I,J} = \sum_{|K|=d-k} M_{I,K} M_{J,K} = \sum_{\substack{|K|=d-k \\ K \cap I = K \cap J = \emptyset}} 1 = \binom{n - |I \cup J|}{d - k}$$

Notice that the value of a diagonal entry  $B_{I,I} = \binom{n-k}{d-k}$  is independent of  $I$ , hence we can easily compute the trace of  $B$ :  $\text{Tr}(B) = \binom{n-k}{d-k} \binom{n}{k}$ . A diagonal entry of  $B^2$  is of the form  $(B^2)_{I,I} = \sum_{|J|=k} (B_{I,J})^2$ . In order to obtain an upper bound on  $v_n$ , it is enough to lower bound  $\text{Tr}(B^2)$ . Since all the terms are non-negative, we will consider the following subsum:

$$(B^2)_{I,I} \geq \sum_{\substack{|J|=k \\ I \cap J = \emptyset}} (B_{I,J})^2 = \sum_{\substack{|J|=k \\ I \cap J = \emptyset}} \binom{n-2k}{d-k}^2 = \binom{n-2k}{d-k}^2 \binom{n-k}{k}.$$

Hence  $\text{Tr}(B^2) = \sum_{|I|=k} (B^2)_{I,I} \geq \binom{n-2k}{d-k}^2 \binom{n-k}{k} \binom{n}{k}$ . Finally, we obtain the following upper bound

$$v_n \leq \frac{\binom{n-k}{d-k}^2 \binom{n}{k}^2}{\binom{n-2k}{d-k}^2 \binom{n-k}{k} \binom{n}{k}} \xrightarrow{n \rightarrow \infty} 1 \tag{9}$$

◀

This proves that for constant  $k, d$ , the gap can be as large as we want, but one can ask whether such large gaps can also be achieved when  $k$  and  $d$  are increasing functions of  $n$ . Let us consider the case where  $k$  and  $d$  are proportional to  $n$ , i.e.,  $k = \alpha n$  and  $d = \beta n$  for some constants  $\alpha, \beta < 1$ . Now, it is no longer true that  $v_n \rightarrow 1$ . For example, for  $\alpha = 0.2$  and

<sup>2</sup> Variations on this matrix also proved useful for the analysis of the *shifted* partial derivatives of symmetric polynomials [5].

$\beta = 0.4$  we have that  $v_n \rightarrow \infty$ . However, we can still prove that  $\frac{v_n}{u_n} \rightarrow 0$  for certain values of  $\alpha$  and  $\beta$ . In the following proposition, to make sure that  $k = \alpha n$  and  $d = \beta n$  are always integers we set  $k = k'm, d = d'm$  and  $n = n'm$  where  $m$  is a new parameter and  $k', d', n'$  are constants (so  $\alpha = k'/n'$  and  $\beta = d'/n'$ ).

► **Proposition 14.** *For any positive integers  $k', d', n'$  such that  $k' < d' < n'/2$ , the family of polynomials  $f_m = \text{Sym}_{d'm, n'm}$  has the following property: if we consider  $u_m = \dim \partial^{=k'm} f_m$  and  $v_m = \frac{(\text{Tr} B_m)^2}{\text{Tr}(B_m^2)}$ , we have  $\frac{v_m}{u_m} \rightarrow 0$ .*

The proof is omitted due to lack of space.

#### 4 #P-hardness result for the space of partial derivatives

In this section it is convenient to work with the space  $\partial^+ f$  spanned by partial derivatives of  $f$  of order  $r$  where  $1 \leq r \leq \deg(f) - 1$ . We will work with homogeneous polynomials, and for those polynomials we have  $\dim \partial^+ f = \dim \partial^* f - 2$ .

► **Theorem 15.** *It is #P-hard to compute  $\dim \partial^* f$  for an input polynomial  $f$  given in expanded form (i.e., written as a sum of monomials). This result remains true for multilinear homogeneous polynomials with coefficients in  $\{0, 1\}$ .*

We proceed by reduction from the problem of counting the number of independent sets in a graph, and use as an intermediate step a problem of topological origin. Recall that an (abstract) simplicial complex is a family  $\Delta$  of subsets of a finite set  $S$  such that for every  $F$  in  $\Delta$ , all the nonempty subsets of  $F$  are also in  $\Delta$ . The elements of  $\Delta$  are also called *faces* of the simplicial complex. We denote by  $|\Delta|$  the number of faces of  $\Delta$ , and more generally by  $|X|$  the cardinality of any finite set  $X$ . The *dimension* of a face  $X \in \Delta$  is  $|X| - 1$ . The dimension of  $\Delta$  is the maximal dimension of its faces. If every face of  $\Delta$  belongs to a face of dimension  $\dim(\Delta)$ , the simplicial complex is said to be *pure*.

The simplicial complex generated by a family  $F_1, \dots, F_m$  of subsets of  $S$  is the smallest simplicial complex containing all of the  $F_i$  as faces. This is simply the family of nonempty subsets  $Y \subseteq S$  such that  $Y \subseteq F_i$  for some  $i$ .

► **Theorem 16.** *The following problem is #P-complete: given a family  $F_1, \dots, F_m$  of subsets of  $[n] = \{1, \dots, n\}$ , compute the number of faces of the simplicial complex  $\Delta$  that it generates. This result remains true if  $\Delta$  is pure, i.e., if  $F_1, \dots, F_m$  have the same cardinality.*

We deduce Theorem 15 from Theorem 16. Let  $\Delta$  be the pure simplicial complex generated by a family  $F_1, \dots, F_m$  of subsets of  $[n]$ , with  $|F_i| = d$  for all  $i$ . We associate to each  $F_i$  the monomial  $m_i = \prod_{j \in F_i} X_j$ , and to  $\Delta$  the polynomial  $f(X_1, \dots, X_n, Y_1, \dots, Y_m) = \sum_{i=1}^m Y_i \cdot m_i(X_1, \dots, X_n)$ . This is a multilinear homogeneous polynomial of degree  $d + 1$  in  $m + n$  variables. Theorem 15 is an immediate consequence of Theorem 16 and of the following lemma.

► **Lemma 17.** *A basis of the linear space spanned by  $\partial^+ f$  consists of the following set of  $2|\Delta|$  polynomials:*

- (i) *The  $|\Delta|$  monomials of the form  $\prod_{j \in F} X_j$ , where  $F$  is a face of  $\Delta$ .*
- (ii) *The  $|\Delta|$  polynomials of the form  $\partial f / \partial F$ , where  $F$  is a face of  $\Delta$  (we denote by  $\partial f / \partial F$  the polynomial obtained from  $f$  by differentiating with respect to all variables  $X_j$  with  $j \in F$ ).*

*In particular,  $\dim(\partial^+ f) = 2|\Delta|$ .*

## 37:12 On the Complexity of Partial Derivatives

**Proof.** We first note that a polynomial (ii) belongs to  $\partial^+ f$  by definition. A polynomial in (i) also belongs to  $\partial^+ f$  since it can be obtained by picking a maximal face  $F_i$  containing  $F$ , differentiating with respect to  $Y_i$ , and then with respect to all variables  $X_j$  where  $j \in F_i \setminus F$ .

Conversely, any partial derivative which is not identically 0 is of the form (i) if we have differentiated  $f$  with respect to exactly one  $Y_i$ , or of the form (ii) if we have not differentiated  $f$  with respect to any of the variables  $Y_i$ . It therefore remains to show that the polynomials in our purported basis are linearly independent.

The monomials in (i) are linearly independent since they are pairwise distinct. To show that the polynomials in (ii) are linearly independent, consider a linear combination

$$g = \sum_{j=1}^{|\Delta|} \alpha_j \frac{\partial f}{\partial G_j},$$

where  $G_1, \dots, G_{|\Delta|}$  are the faces of  $\Delta$ . By construction of  $f$ ,

$$g = \sum_{i=1}^m Y_i \cdot \left( \sum_{j=1}^{|\Delta|} \alpha_j \frac{\partial m_i}{\partial G_j} \right). \quad (10)$$

Assume that some coefficient  $\alpha_j$ , for instance  $\alpha_1$ , is different from 0. The face  $G_1$  belongs to some maximal face of  $\Delta$ , for instance to  $F_1$ . We claim that  $\sum_{j=1}^{|\Delta|} \alpha_j \frac{\partial m_1}{\partial G_j} \neq 0$ . Indeed, the faces  $G_j$  which are not included in  $F_1$  contribute nothing to this sum, and the faces that are included in  $F_1$  contribute pairwise distinct monomials. It follows from (10) that  $g \neq 0$ , and that the polynomials in (ii) are indeed linearly independent.

To complete the proof of the lemma, it remains to note that the spaces spanned by (i) and (ii) are in direct sum. Indeed, the first space is included in  $\mathbb{Q}[X_1, \dots, X_n]$  while the second is included in  $\sum_{i=1}^m Y_i \mathbb{Q}[X_1, \dots, X_n]$ .  $\blacktriangleleft$

### 4.1 Proof of Theorem 16

Let  $G = (V, E)$  be a graph with vertex set  $V = [n]$  and  $m = |E|$  edges. We associate to  $G$  the simplicial complex  $\Delta$  generated by the complements of the edges of  $G$ , i.e., by the sets  $V \setminus \{u, v\}$  where  $uv \in E$ . This is a pure simplicial complex of dimension  $n - 2$ . The faces of  $\Delta$  are the complements of the dependent sets of  $G$ . Hence  $|\Delta| = 2^n - \text{Ind}(G)$ , where  $\text{Ind}(G)$  denotes the number of independent sets in  $G$ . Computing the number of independent sets of a graph is a well-known  $\#\text{P}$ -complete problem. It was shown to be  $\#\text{P}$ -complete even for bipartite graphs [15], for planar bipartite graphs of degree at most four [19] and for 3-regular graphs [7]. It follows that computing  $|\Delta|$  is  $\#\text{P}$ -hard, and membership in  $\#\text{P}$  is immediate from the definition. This completes the proof of Theorem 16.

We note that it is easy to shortcut Theorem 16 and construct the polynomial  $f$  in the proof of Theorem 15 directly from  $G$ : we have

$$f = \sum_{uv \in E} Y_{uv} \cdot \prod_{w \notin \{u, v\}} X_w$$

and  $\dim \partial^+ f = 2(2^n - \text{Ind}(G))$ . Since the maximal faces of  $\Delta$  have  $n - 2$  elements, we have the following refinement of Theorem 15.

**► Corollary 18.** *It is  $\#\text{P}$ -hard to compute  $\dim \partial^* f$  for a multilinear homogenous polynomial  $f$  of degree  $n - 1$  with coefficients in  $\{0, 1\}$ ,  $m$  monomials and  $n + m$  variables with  $m \leq \binom{n}{2}$ .*

---

**References**

---

- 1 Noga Alon. Perturbed identity matrices have high rank: Proof and applications. *Combinatorics, Probability and Computing*, 18(1-2):3–15, 2009.
- 2 Xi Chen, Neeraj Kayal, and Avi Wigderson. Partial derivatives in arithmetic complexity and beyond. *Found. Trends Theor. Comput. Sci.*, 6(1-2):front matter, 1–138 (2011), 2010.
- 3 Martin Dyer, Alan Frieze, and Mark Jerrum. On counting independent sets in sparse graphs. *SIAM J. Comput.*, 31(5):1527–1541, 2002.
- 4 Martin Dyer and Catherine Greenhill. On Markov chains for independent sets. *J. Algorithms*, 35(1):17–49, 2000.
- 5 Hervé Fournier, Nutan Limaye, Meena Mahajan, and Srikanth Srinivasan. The shifted partial derivative complexity of elementary symmetric polynomials. In *Mathematical foundations of computer science 2015. Part II*, volume 9235 of *Lecture Notes in Comput. Sci.*, pages 324–335. Springer, Heidelberg, 2015.
- 6 D. H. Gottlieb. A certain class of incidence matrices. *Proc. Amer. Math. Soc.*, 17:1233–1237, 1966.
- 7 Catherine Greenhill. The complexity of counting colourings and independent sets in sparse graphs and hypergraphs. *Comput. Complexity*, 9(1):52–72, 2000.
- 8 Christopher J. Hillar and Lek-Heng Lim. Most tensor problems are NP-hard. *J. ACM*, 60(6):Art. 45, 39, 2013.
- 9 Neeraj Kayal. Affine projections of polynomials. In *STOC'12 – Proceedings of the 2012 ACM Symposium on Theory of Computing*, pages 643–661. ACM, New York, 2012.
- 10 Neeraj Kayal, Nutan Limaye, Saha Chiranjib, and Sudarshan Srinivasan. An exponential lower bound for homogeneous depth four arithmetic formulas. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 61–70. IEEE, 2004.
- 11 Neeraj Kayal and Chandan Saha. Lower bounds for depth three arithmetic circuits with small bottom fanin. In *30th Conference on Computational Complexity*, volume 33 of *LIPICs. Leibniz Int. Proc. Inform.*, pages 158–182. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2015.
- 12 Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, Cambridge, 1997.
- 13 Michael Luby and Eric Vigoda. Approximately counting up to four. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 682–687. ACM, 1997.
- 14 Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Comput. Complexity*, 6(3):217–234, 1996/97.
- 15 J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4):777–788, 1983.
- 16 Bjarke Hammersholt Roune and Eduardo Sáenz-de Cabezón. Complexity and algorithms for Euler characteristic of simplicial complexes. *J. Symbolic Comput.*, 50:170–196, 2013.
- 17 Ramprasad Saptharishi. A survey of lower bounds in arithmetic circuit complexity. URL: <https://github.com/dasarpmar/lowerbounds-survey/releases>.
- 18 Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: a survey of recent results and open questions. *Found. Trends Theor. Comput. Sci.*, 5(3-4):207–388, 2010.
- 19 Salil P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM J. Comput.*, 31(2):398–427 (electronic), 2001.
- 20 L. G. Valiant. The complexity of computing the permanent. *Theoret. Comput. Sci.*, 8(2):189–201, 1979.



# Set Membership with Non-Adaptive Bit Probes

Mohit Garg<sup>\*1</sup> and Jaikumar Radhakrishnan<sup>2</sup>

1 Tokyo Institute of Technology, Tokyo, Japan

[garg.m.aa@m.titech.ac.jp](mailto:garg.m.aa@m.titech.ac.jp)

2 Tata Institute of Fundamental Research, Mumbai, India

[jaikumar@tifr.res.in](mailto:jaikumar@tifr.res.in)

---

## Abstract

We consider the non-adaptive bit-probe complexity of the set membership problem, where a set  $S$  of size at most  $n$  from a universe of size  $m$  is to be represented as a short bit vector in order to answer membership queries of the form “Is  $x$  in  $S$ ?” by *non-adaptively* probing the bit vector at  $t$  places. Let  $s_N(m, n, t)$  be the minimum number of bits of storage needed for such a scheme. Buhrman, Miltersen, Radhakrishnan, and Srinivasan [4] and Alon and Feige [1] investigated  $s_N(m, n, t)$  for various ranges of the parameter  $t$ . We show the following.

*General upper bound* ( $t \geq 5$  and odd): For odd  $t \geq 5$ ,  $s_N(m, n, t) = O(tm^{\frac{2}{t-1}}n^{1-\frac{2}{t-1}}\lg\frac{2m}{n})$ . This improves on a result of Buhrman *et al.* that states for odd  $t \geq 5$ ,  $s_N(m, n, t) = O(m^{\frac{4}{t+1}}n)$ . For small values of  $t$  (odd  $t \geq 3$  and  $t \leq \frac{1}{10}\lg\lg m$ ) and  $n \leq m^{1-\epsilon}$  ( $\epsilon > 0$ ), we obtain adaptive schemes that use a little less space:  $O(\exp(e^{2t})m^{\frac{2}{t+1}}n^{1-\frac{2}{t+1}}\lg m)$ .

*Three probes* ( $t = 3$ ) *lower bound*: We show that  $s_N(m, n, 3) = \Omega(\sqrt{mn})$  for  $n \geq n_0$  for some constant  $n_0$ . This improves on a result of Alon and Feige that states that for  $n \geq 16\lg m$ ,  $s_N(m, n, 3) = \Omega(\sqrt{\frac{mn}{\lg m}})$ . The complexity of the non-adaptive scheme might, in principle, depend on the function that is used to determine the answer based on the three bits read (one may assume that all queries use the same function). Let  $s_N^f(m, n, 3)$  be the minimum number of bits of storage required in a three-probe non-adaptive scheme where the function  $f : \{0, 1\}^3 \rightarrow \{0, 1\}$  is used to answer the queries. We show that for large class of functions  $f$  (including the majority function on three bits), we in fact have  $s_N(m, n, 3) = \Omega(m^{1-\frac{1}{cn}})$  for  $n \geq 4$  and some  $c > 0$ . In particular, three-probe non-adaptive schemes that use such query functions  $f$  do not give any asymptotic savings over the trivial characteristic vector when  $n \geq \log m$ .

**1998 ACM Subject Classification** E.1 Data Structures, E.4 Coding and Information Theory

**Keywords and phrases** Data Structures, Bit-probe model, Compression, Bloom filters, Expansion

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.38

## 1 Introduction

The *set membership problem* is a fundamental problem in the area of data structures and information compression and retrieval. In its abstract form we are given a subset  $S$  of size at most  $n$  from a universe of size  $m$  and required to represent it as a bit string so that membership queries of the form “Is  $x$  in  $S$ ?” can be answered using a small number of probes into the bit string. The characteristic function representation provides a solution to this problem: just one bit-probe is needed to answer queries, but all sets are represented using  $m$ -bit strings (which is very wasteful when  $n$  is promised to be small).

---

\* A part of this work was done when the first author was at the Tata Institute of Fundamental Research.



The trade-off between the number of bits in the representation and the number of probes is the subject of several previous works: it was studied by Minsky and Papert in their 1969 book *Perceptrons* [11]; more recently, Buhrman, Miltersen, Radhakrishnan and Venkatesh [4] showed the existence of randomized schemes that answer queries with just one bit probe and use near optimal space. In contrast, they showed that deterministic schemes that answer queries by making a constant number of probes cannot use optimal space. The deterministic worst-case trade-off for this problem was also considered in the same paper and in several subsequent works (e.g., Radhakrishnan, Raman and Rao [12], Alon and Feige [1], Radhakrishnan, Shah and Shannigrahi [14], Viola [15], Lewenstein, Munro, Nicholson and Raman [8], Garg and Radhakrishnan [5]). For sets where each element is included with probability  $p$ , Makhdoumi, Huang, Médard and Polyanskiy [10] showed, in particular, that no savings over the characteristic vector can be obtained in this case for non-adaptive schemes with  $t = 2$ .

In this work, we focus on deterministic schemes with *non-adaptive* probes, where the probes are made in parallel (or equivalently the location of probes do not depend on the value read in previous probes). Such schemes have been studied in several previous works. Let  $s_N(m, n, t)$  be the minimum number of bits of storage required in order to answer membership queries with  $t$  non-adaptive probes.

► **Definition 1.** A non-adaptive  $(m, n, s, t)$ -scheme consists of a storage function and a query scheme. The storage function has the form  $\phi : \binom{[m]}{\leq n} \rightarrow \{0, 1\}^s$  that takes a set of size at most  $n$  and returns its  $s$ -bit representation. The query scheme associates with each element  $x$  the  $t$  probe locations  $(i_1(x), \dots, i_t(x)) \in [s]^t$  and a function  $f_x : \{0, 1\}^t \rightarrow \{0, 1\}$ . We require that for all  $S \in \binom{[m]}{\leq n}$  and all  $x \in [m]$ :  $x \in S$  iff  $f_x(\phi(S)[i_1(x)], \phi(S)[i_2(x)], \dots, \phi(S)[i_t(x)]) = 1$ . Let  $s_N(m, n, t)$  denote the minimum  $s$  such that there is an  $(m, n, s, t)$ -scheme.

In our discussion, we use  $s(m, n, t)$  (without the subscript  $N$ ) to denote the minimum space required for adaptive schemes. Using the above notation, we now describe our results and their relation to what was known before. All asymptotic claims below hold for large  $m$ .

## 1.1 General non-adaptive schemes

► **Theorem 2** (Result 1, non-adaptive schemes). *For odd  $t \geq 5$ , we have*

$$s_N(m, n, t) = O(tm^{\frac{2}{t-1}} n^{1-\frac{2}{t-1}} \lg \frac{2m}{n}).$$

In comparison, for odd  $t \geq 5$ , Buhrman *et al.* showed that  $s_N(m, n, t) = O(m^{\frac{4}{t+1}} n)$ . The exponent of  $m$  in their upper bound result is roughly four times the exponent of  $m$  appearing in their lower bound result. Their schemes are non-adaptive and use the MAJORITY function to answer membership queries. We exhibit schemes that still use MAJORITY but need less space. Buhrman *et al.* also show a lower bound of  $s(m, n, t) = \Omega(tm^{\frac{1}{t}} n^{1-\frac{1}{t}})$  valid (even for adaptive schemes) when  $n \leq m^{1-\epsilon}$  (for  $\epsilon > 0$  and  $t \ll \lg m$ ). Note that the exponent of  $m$  in our result is twice the exponent of  $m$  appearing in the lower bound result. These schemes, as well as the non-adaptive scheme for  $t = 4$  due to Alon and Feige [1], have implications for the problem studied by Makhdoumi *et al.* [10]; unlike in the case of  $t = 2$ , significant savings are possible if  $t \geq 4$ , even with non-adaptive schemes<sup>1</sup>. Using a similar proof idea, we obtain slightly better upper bound with adaptive schemes when  $t$  is small and  $n$  is at most  $m^{1-\epsilon}$ .

<sup>1</sup> We are grateful to Tom Courtade and Ashwin Pananjady for this observation.



► **Theorem 3** (Result 2, adaptive schemes). *For odd  $t \geq 3$  and  $t \leq \frac{1}{10} \lg \lg m$  and for  $n \leq m^{1-\epsilon}$  ( $\epsilon > 0$ ), we have  $s(m, n, t) = O(\exp(e^{2t}) m^{\frac{2}{t+1}} n^{1-\frac{2}{t+1}} \lg m)$ .*

*Technique.* To justify our claim, we need to describe the query scheme, that is,  $(i_1(x), i_2(x), \dots, i_t(x))$  for each  $x \in [m]$  and the query function  $f_x : \{0, 1\}^t \rightarrow \{0, 1\}$ . For  $f_x$  we use the MAJORITY on  $t$  bits ( $t$  is odd). The locations to be probed for each element will be obtained using a probabilistic argument. Once a query scheme is fixed, we need to show how the assignment to the memory is obtained. For this, we describe a sequential algorithm. We show that the random assignment of locations ensures sufficient expansion allowing us to start with a greedy argument arrange that most queries are answered correctly, and then use Hall's bipartite graph matching theorem to find the required assignment for the remaining elements. Versions of this argument have been used in previous works [9, 4, 7, 1, 5].

## 1.2 Three non-adaptive probes

For one probe and  $m \geq 2$ , it is easy to show that no space can be saved over the characteristic vector representation. For two non-adaptive probes, only for the special case  $n = 1$ , some non-trivial savings over the characteristic vector representation are possible:  $s_N(m, 1, 2) = \theta(\sqrt{m})$ . For  $n \geq 2$ , Buhrman *et al.* [4] showed  $s_N(m, n, 2) = m$ . The smallest number of probes for which the complexity of problem with non-adaptive probes is not settled is three. Observe that any scheme with two adaptive probes can be converted to a scheme with three non-adaptive probes; the two probe decision tree has at most three nodes. Thus, using the two adaptive probes upper bound result of Garg and Radhakrishnan [5], we have  $s_N(m, n, 3) \leq s(m, n, 2) = O(m^{1-\frac{1}{4n+1}})$ . Thus, non-trivial savings in space over the characteristic vector representation is possible when  $n = o(\lg m)$ . Consequently, the question is whether more space can be saved or is this upper bound tight? We are not aware of any three-probe non-adaptive scheme that manages with  $o(m)$  space for sets of size  $\omega(\lg m)$ . Alon and Feige [1] show the following lower bound:  $s_N(m, n, 3) = \Omega(\sqrt{\frac{mn}{\lg m}})$  for  $n \geq 16 \lg m$ .

In order to obtain better lower bounds for three-probe non-adaptive schemes, we proceed as follows. In any three-probe non-adaptive scheme, the query scheme specifies, for each element, the three locations to probe and a three variable boolean function to be applied on three values read. In principle, for different elements, the query scheme can specify different boolean functions. But since there are only a finite number (256) of boolean functions on three variables, some set of at least  $m/256$  elements of the universe use a common function. We may thus restrict attention to this part of the universe, and assume that the function being employed to answer queries is always the same. Furthermore, we may place functions obtained from one another by negating and permuting variables in a common equivalence class, and restrict our attention to one representative in each class. For three variable boolean functions, Pólya counting yields that there are twenty-two equivalence classes. This classification of the 256 functions into twenty-two classes is already available in the literature [16]. We show the following.

► **Theorem 4** (Result 3).

- (a) *If the query function  $f : \{0, 1\}^3 \rightarrow \{0, 1\}$  is not equivalent to  $(x, y, z) \mapsto (x \wedge y) \oplus z$  or  $(x, y, z) \mapsto 1$  iff  $x + y + z = 1$ , then  $s_N(m, n, 3) = \Omega(m^{1-\frac{1}{cn}})$  for  $n \geq 4$  and some  $c > 0$ .*
- (b) *If the query function  $f : \{0, 1\}^3 \rightarrow \{0, 1\}$  is equivalent to  $(x, y, z) \mapsto (x \wedge y) \oplus z$  or  $(x, y, z) \mapsto 1$  iff  $x + y + z = 1$ , then  $s_N(m, n, 3) = \Omega(\sqrt{mn})$ .*
- (c) *If the query function  $f : \{0, 1\}^3 \rightarrow \{0, 1\}$  is equivalent to  $(x, y, z) \mapsto (x \wedge y) \oplus z$  and  $\lg m \leq n \leq \frac{m}{\lg m}$ , then  $s_N(m, n, 3) = \Omega(\sqrt{mn \frac{\lg \frac{m}{n}}{\lg \lg m}})$ .*

The best upper bounds for non-adaptive schemes with four or more probes use the MAJORITY function to answer membership queries. Our result implies that for three non-adaptive probes, when queries are answered by computing MAJORITY, the space required is at least  $\Omega(m^{1-\frac{1}{cn}})$  for some constant  $c$ . In fact, similar lower bound holds if membership queries are answered using most boolean functions. Our results do not yield a similar lower bound for  $(x, y, z) \mapsto (x \wedge y) \oplus z$  and  $(x, y, z) \mapsto 1$  iff  $x + y + z = 1$  types. For these two types of query functions, we get a slightly better lower bound than what is implied by [1]. Thus, further investigations on three probes non-adaptive schemes need to focus on just  $(x, y, z) \mapsto 1$  iff  $x + y + z = 1$  and  $(x, y, z) \mapsto (x \wedge y) \oplus z$  as the query functions.

*Technique.* As mentioned above, there are twenty-two types of functions for which we need to prove a lower bound. Seven of the twenty-two classes contain functions that can be represented by a decision tree of height at most two. Thus, for these functions, the two probe adaptive lower bound in [5] implies the result. These functions are: constant 0, constant 1, the DICTATOR function  $(x, y, z) \mapsto x$ , the function  $(x, y, z) \mapsto x \wedge y$ , its complement  $(x, y, z) \mapsto \bar{x} \vee \bar{y}$ ,  $(x, y, z) \mapsto (x \wedge y) \vee (\bar{x} \wedge \bar{z})$ , and  $(x, y, z) \mapsto (x \wedge y) \vee (\bar{x} \wedge \bar{y})$ .

After this, fifteen classes remain. Functions in some eleven of the remaining fifteen classes admit a density argument, similar in spirit to the adaptive two-probes lower-bound proof in [5]. To streamline the argument, we classify these eleven classes into two parts. The first part contains the MAJORITY function. The second part contains the AND function, the ALL-EQUAL function, the functions  $(x, y, z) \mapsto (x \oplus y) \wedge z$ ,  $(x, y, z) \mapsto (x \vee y) \wedge z$ ,  $(x, y, z) \mapsto (x \wedge y \wedge z) \vee (\bar{y} \wedge \bar{z})$ , and their complements. For functions in the second part we deal with two functions—a function and its complement—with a single proof. In these proofs, we produce sets  $S$  and  $T$  of size at most  $n$  such that storing  $S$  and not storing  $T$  leads to a contradiction. The proof for the complement function works with a small twist: storing  $T$  and not storing  $S$  leads to the contradiction. Thus, these eleven cases are handled by six proofs. In each of these proofs we roughly argue (sometimes probabilistically) that if the scheme is valid, it must conceal a certain dense graph that avoids small cycles. Standard graph theoretic results (the Moore bound) that relate density and girth then gives us the lower bound.

For the remaining four classes, we employ linear-algebraic arguments. Representatives chosen from these classes are PARITY,  $(x, y, z) \mapsto 1$  iff  $x + y + z \neq 1$ ,  $(x, y, z) \mapsto (x \wedge y) \oplus z$ , and  $(x, y, z) \mapsto 1$  iff  $x + y + z = 1$ . For PARITY and  $(x, y, z) \mapsto 1$  iff  $x + y + z \neq 1$ , we show using standard dimension argument, that if the space used is smaller than the universe size  $m$ , then there is some element  $u \in [m]$  that is (linearly) dependent on the other elements. Not storing the other elements, leaves the scheme with no choice for  $u$ , thus leading to a contradiction. For  $(x, y, z) \mapsto (x \wedge y) \oplus z$  and  $(x, y, z) \mapsto 1$  iff  $x + y + z = 1$  a modification of an algebraic argument of Radhakrishnan, Sen and Venkatesh [13] implies a lower bound of  $\sqrt{mn}$ . (Interestingly, we need to choose an appropriate characteristic of the field (2 or 3) based on which function we deal with.) For  $(x, y, z) \mapsto (x \wedge y) \oplus z$ , we further improve on this argument by employing random restrictions. These results together improve the previous best lower bound (due to Alon and Feige [1]) irrespective of the query function used.

## 2 General non-adaptive upper bound

In this section, we prove the general non-adaptive upper bound result: Theorem 2.

► **Definition 5.** A non-adaptive  $(m, s, t)$ -graph is a bipartite graph  $G$  with vertex sets  $U = [m]$  and  $V$  ( $|V| = ts$ ).  $V$  is partitioned into  $t$  disjoint sets:  $V_1, \dots, V_t$ ; each  $V_i$  has  $s$  vertices. Every  $u \in U$  has a unique neighbour in each  $V_i$ . A non-adaptive  $(m, s, t)$ -graph

naturally gives rise to a non-adaptive  $(m, ts, t)$ -query scheme  $\mathcal{T}_G$  as follows. We view the memory (an array  $L$  of  $ts$  bits) to be indexed by vertices in  $V$ . On receiving the query “Is  $u$  in  $S$ ?”, we answer “Yes” iff the MAJORITY of the locations in the neighbourhood of  $u$  contain a 1. We say that the query scheme  $\mathcal{T}_G$  is satisfiable for a set  $S \subseteq [m]$ , if there is an assignment to the memory locations  $(L[v] : v \in V)$ , such that  $\mathcal{T}_G$  correctly answers all queries of the form “Is  $x$  in  $S$ ?”.

We now restrict attention to odd  $t \geq 5$ . First, we identify an appropriate property of the underlying non-adaptive  $(m, s, t)$ -graph  $G$  that guarantees that  $\mathcal{T}_G$  is satisfiable for all sets  $S$  of size at most  $n$ . We then show that such a graph exists for some  $s = O(m^{\frac{2}{t-1}} n^{1-\frac{2}{t-1}} \lg \frac{2m}{n})$ .

► **Definition 6** (Non-adaptive admissible graph). We say that a non-adaptive  $(m, s, t)$ -graph  $G$  is admissible for sets of size at most  $n$  if the following two properties hold:

(P1)  $\forall R \subseteq [m]$  ( $|R| \leq n + \lceil 2n \lg \frac{2m}{n} \rceil$ ):  $|\Gamma_G(R)| \geq \frac{t+1}{2}|R|$ , where  $\Gamma_G(R)$  is the set of neighbors of  $R$  in  $G$ .

(P2)  $\forall S \subseteq [m]$  ( $|S| = n$ ):  $|T_S| \leq \lceil 2n \lg \frac{2m}{n} \rceil$ , where  $T_S = \{y \in [m] \setminus S : |\Gamma_G(y) \cap \Gamma_G(S)| \geq \frac{t+1}{2}\}$ .

Our theorem will follow from the following claims.

► **Lemma 7.** *If a non-adaptive  $(m, s, t)$ -graph  $G$  is admissible for sets of size at most  $n$ , then the non-adaptive  $(m, ts, t)$ -query scheme  $\mathcal{T}_G$  is satisfiable for every set  $S$  of size at most  $n$ .*

► **Lemma 8.** *There is a non-adaptive  $(m, s, t)$ -graph, with  $s = O(m^{\frac{2}{t-1}} n^{1-\frac{2}{t-1}} \lg \frac{2m}{n})$ , that is admissible for every set  $S \subseteq [m]$  of size at most  $n$ .*

**Proof of Lemma 7.** Fix an admissible graph  $G$ . Thus,  $G$  satisfies (P1) and (P2) above. Fix a set  $S \subseteq [m]$  of size at most  $n$ . We will show that there is a 0-1 assignment to the memory such that all queries are answered correctly by  $\mathcal{T}_G$ .

Let  $S' \subseteq [m]$  be such that  $S \subseteq S'$  and  $|S'| = n$ . From (P2), we know  $|T_{S'}| \leq \lceil 2n \lg \frac{2m}{n} \rceil$ . Hence,  $|S' \cup T_{S'}| \leq n + \lceil 2n \lg \frac{2m}{n} \rceil$ . From (P1) and Hall’s theorem, we may assign to each element  $u \in S' \cup T_{S'}$  a set  $A_u \subseteq V$  such that (i)  $|A_u| = \frac{t+1}{2}$  and (ii) the  $A_u$ ’s are disjoint. For each  $u \in S \subseteq S'$ , we assign the value 1 to all locations in  $A_u$ . For each  $u \in (S' \cup T_{S'}) \setminus S$ , we assign the value 0 to all locations in  $A_u$ . Since  $\frac{t+1}{2} > \frac{t}{2}$ , all queries for  $u \in S' \cup T_{S'}$  are answered correctly.

Assign 0 to all locations in  $\Gamma_G([m] \setminus (S' \cup T_{S'}))$ . For  $y \in [m] \setminus (S' \cup T_{S'})$ ,  $|\Gamma_G(y) \cap \Gamma_G(S)| \leq \frac{t-1}{2}$ . As a result, queries for elements in  $[m] \setminus (S' \cup T_{S'})$  are answered correctly, as the majority evaluates to 0 for each one of them. ◀

**Proof of Lemma 8.** In the following, set

$$s = \left\lceil 60m^{\frac{2}{t-1}} n^{1-\frac{2}{t-1}} \lg \frac{2m}{n} \right\rceil.$$

We show that a suitable random non-adaptive  $(m, s, t)$ -graph  $G$  is admissible for sets of size at most  $n$  with positive probability. The graph  $G$  is constructed as follows. Recall that  $V = \bigcup_i V_i$ . For each  $u \in U$ , one neighbor is chosen uniformly and independently in each  $V_i$ .

(P1) holds. If (P1) fails, then for some non-empty  $W \subseteq U$ , ( $|W| \leq n + \lceil 2n \lg \frac{2m}{n} \rceil$ ), we have  $|\Gamma_G(W)| \leq \frac{t+1}{2}|W| - 1$ . Fix a set  $W$  of size  $r \geq 1$  and  $L \subseteq V$  of size  $\frac{t+1}{2}r - 1$ . Let  $L$  have  $\ell_i$  elements in  $V_i$ ; thus,  $\sum_i \ell_i = \frac{t+1}{2}r - 1$ . Then,

$$\Pr[\Gamma_G(W) \subseteq L] \leq \prod_{i=1}^t \left( \frac{\ell_i}{|V_i|} \right)^r \leq \left( \frac{\left(\frac{t+1}{2}\right)r - 1}{ts} \right)^{tr},$$

where the last inequality is a consequence of  $GM \leq AM$ . We conclude, using the union bound over choices of  $W$  and  $L$ , that (P1) fails with probability at most

$$\sum_{r=1}^{n + \lceil 2n \lg \frac{2m}{n} \rceil} \binom{m}{r} \binom{ts}{\frac{t+1}{2}r - 1} \left( \frac{\frac{t+1}{2}r - 1}{ts} \right)^{tr} \quad (1)$$

$$\begin{aligned} &\leq \sum_{r=1}^{n + \lceil 2n \lg \frac{2m}{n} \rceil} \left( \frac{em}{r} \right)^r \left( \frac{tes}{\frac{t+1}{2}r - 1} \right)^{\frac{t+1}{2}r - 1} \left( \frac{\frac{t+1}{2}r - 1}{ts} \right)^{tr} \\ &\leq \sum_{r=1}^{n + \lceil 2n \lg \frac{2m}{n} \rceil} \left[ \frac{(e^{\frac{t+3}{2} - \frac{1}{r}})mr^{\frac{t-1}{2} - 1 + \frac{1}{r}}}{(s^{\frac{1}{r}})s^{\frac{t-1}{2}}} \right]^r \leq \frac{1}{3}, \end{aligned} \quad (2)$$

where the last inequality holds because we have chosen  $s$  large enough.

**(P2) holds.** For (P2) to fail, there must exist a set  $S \subseteq [m]$  of size  $n$  such that  $|T_S| > \lceil 2n \lg \frac{2m}{n} \rceil$ . Fix a set  $S$  of size  $n$ . Fix a  $y \in [m] \setminus S$ .

$$\Pr[y \in T_S] \leq \binom{t}{\frac{t+1}{2}} \left( \frac{n}{s} \right)^{\frac{t+1}{2}} \leq \frac{n}{10m},$$

where the last inequality holds because of choice of  $s$  and  $m$  is large. Thus,  $\mathbb{E}[|T_S|] \leq \frac{n}{10}$ . To conclude that  $|T_S|$  is bounded with high probability, we will use the following version of Chernoff bound: if  $X = \sum_{i=1}^N X_i$ , where each random variable  $X_i \in \{0, 1\}$  independently, then if  $\gamma > 2e\mathbb{E}[X]$ , then  $\Pr[X > \gamma] \leq 2^{-\gamma}$ . Then, for all large  $m$ ,

$$\Pr[|T_S| > 2n \lg \frac{2m}{n}] \leq 2^{-2n \lg \frac{2m}{n}}.$$

Using the union bound, we conclude that

$$\Pr[(P2) \text{ fails}] \leq \left( \frac{em}{n} \right)^n 2^{-2n \lg \frac{2m}{n}} \leq \frac{1}{3}.$$

Thus, with probability at least  $\frac{1}{3}$  the random graph  $G$  is admissible.  $\blacktriangleleft$

### 3 Three non-adaptive probes lower bound

In this section, we prove the three probe lower bound result: Theorem 4.

► **Definition 9 (Equivalent).** Two boolean functions are called equivalent if one can be obtained from the other by negating and permuting the variables.

► **Proposition 10.** *Let  $f, g : \{0, 1\}^t \rightarrow \{0, 1\}$  be equivalent. If  $s_1$  and  $s_2$  are the minimum bits of space required for non-adaptive  $(m, n, s_1, t)$  and  $(m, n, s_2, t)$ -schemes with query functions  $f$  and  $g$  respectively, then  $s_1 = s_2$ .*

For three variable boolean functions, there are twenty-two equivalence classes (see [16]). To prove Theorem 4, we provide proofs for these twenty-two query functions, each from a different class. In many proofs below we assume that the memory consists of three arrays of size  $s$  each, and the three probes are made on different arrays. Given any scheme that uses space  $s$ , we can always modify it to meet our assumption, by expanding the space by factor 3.

### 3.1 Decision trees of height two

Seven of the twenty-two classes contain functions that can be represented by a decision tree of height at most two. Thus, for these functions, the two probe adaptive lower bound [5] implies the result. These functions are: constant 0, constant 1, the DICTATOR function  $(x, y, z) \mapsto x$ , the function  $(x, y, z) \mapsto x \wedge y$ , its complement  $(x, y, z) \mapsto \bar{x} \vee \bar{y}$ ,  $(x, y, z) \mapsto (x \wedge y) \vee (\bar{x} \wedge \bar{z})$ , and  $(x, y, z) \mapsto (x \wedge y) \vee (\bar{x} \wedge \bar{y})$ .

### 3.2 MAJORITY

Let  $\Phi$  be a non-adaptive  $(m, n, s, 3)$ -scheme with MAJORITY as the query function. The memory is a bit array  $A[1, \dots, s]$  of length  $s$ . For each element  $u \in [m]$ ,  $x(u), y(u), z(u) \in [s]$  are the three distinct locations in  $A$  that are probed to determine whether  $u$  is in the set or not. For each set  $S \subseteq [m]$  of size at most  $n$ , the assignment  $\sigma(S) \in \{0, 1\}^s$  to  $A$  is such that for all elements  $u \in [m]$ ,  $\text{Maj}(A[x(u)], A[y(u)], A[z(u)])$  is 1 iff  $u \in S$ , where  $\text{Maj}$  is the MAJORITY of 3 bits.

► **Definition 11.** (model-graph for  $\Phi$ , third vertex, meet) Let  $\Phi$  be a  $(m, n, s, 3)$ -scheme with MAJORITY as the query function. Fix a graph  $G$  such that  $V(G) = [s]$ ,  $|E(G)| = m$  and edge labels:  $\{\text{lab}(e) | e \in E(G)\} = [m]$  (there is a unique edge for each label in  $[m]$ ).  $G$  is called a *model-graph* for  $\Phi$  if for each  $u \in [m]$  the edge labelled  $u$  has the set of endpoints in  $\{\{x(u), y(u)\}, \{y(u), z(u)\}, \{z(u), x(u)\}\}$ . For example, the graph  $G = ([s], \{x(u) \xleftarrow{u} y(u) | u \in [m]\})$  is a model graph for  $\Phi$ .

In a model-graph for  $\Phi$ , let  $e$  be the set of endpoints of the edge with label  $u$ . The element in the singleton  $(\{x(u), y(u), z(u)\} \setminus e)$  is defined to be the *third vertex* of  $u$ .

Two edge-disjoint cycles  $C_1$  and  $C_2$  are said to *meet* in a model-graph for  $\Phi$  if there exist elements  $u, v \in [m]$  such that the third vertices of  $u$  and  $v$  are the same vertex and the edges labelled  $u$  and  $v$  are in the different cycles  $C_1$  and  $C_2$  respectively.

► **Definition 12.** A model-graph  $G$  for an  $(m, n, s, 3)$ -scheme with MAJORITY as the query function is said to be *forced* if at least one of the following three conditions hold.

(P1)  $\exists$  edge-disjoint odd cycles  $C_1, C_2$  in  $G$  with lengths at most  $n$  each that intersect at a vertex.

(P2)  $\exists$  edge disjoint even cycles  $C_1, C_2$  in  $G$  with lengths at most  $n$  each and  $C_1$  and  $C_2$  meet.

(P3)  $\exists$  an even cycle  $C$  of length at most  $n$ , such that some two edges in  $C$ , labelled  $e$  and  $f$  say, have an even number of edges between them (while traversing the edges of the cycle in order) and the third vertices of  $e$  and  $f$  are the same vertex.

► **Lemma 13.** A model-graph for a scheme with MAJORITY as the query function cannot be forced.

► **Lemma 14.** Any  $(m, n, \left\lceil \frac{1}{6}m^{1 - \frac{1}{\lfloor \frac{n}{2} \rfloor + 1}} \right\rceil, 3)$ -scheme with MAJORITY as the query function has a forced model-graph.

From lemmas 13 and 14, it follows that when MAJORITY is used as the query function,  $s_N(m, n, 3) > \frac{1}{6}m^{1 - \frac{1}{\lfloor \frac{n}{2} \rfloor + 1}}$ .

**Proof of Lemma 13.** Fix a  $(m, n, s, 3)$ -scheme  $\Phi$  with MAJORITY as the query function. Fix a model-graph  $G$  for  $\Phi$ . Assume  $G$  is forced, that is, it satisfies (P1) or (P2) or (P3) above.

**Case: (P1) holds.** (P1) implies that there are edge-disjoint cycles  $C_1$  and  $C_2$  in  $G$  such that,

$$C_1 : u_0 \xrightarrow{e_1} u_1 \xrightarrow{e_2} \cdots \xrightarrow{e_{2k+1}} u_{2k+1} = u_0,$$

$$C_2 : u_0 \xrightarrow{f_1} v_1 \xrightarrow{f_2} \cdots \xrightarrow{f_{2l+1}} v_{2l+1} = u_0,$$

and  $2k + 1, 2l + 1 \leq n$ . Let  $S_0 = \{e_1, e_3, \dots, e_{2k+1}\} \cup \{f_2, f_4, \dots, f_{2l}\}$  and  $S_1 = \{e_2, e_4, \dots, e_{2k}\} \cup \{f_1, f_3, \dots, f_{2l+1}\}$ . Note that  $|S_0| = |S_1| \leq n$ . We claim that  $\Phi$  cannot represent any set  $S$  such that

$$S_0 \subseteq S \subseteq \bar{S}_1.$$

In particular,  $\Phi$  cannot represent the set  $S_0$ . Assume  $\Phi$  represents such an  $S$ . We claim that  $u_0$  cannot be assigned a 0. If  $u_0$  is assigned a 0, then since  $e_1 \in S$ ,  $u_1$  must be assigned a 1. Otherwise,  $\text{Maj}(A[x(e_1)], A[y(e_1)], A[z(e_1)]) = \text{Maj}(0, 0, b) = 0$ , where  $b$  is the bit assigned to the location in  $\{x(e_1), y(e_1), z(e_1)\} \setminus \{u_0, u_1\}$ . Since,  $u_1$  is assigned a 1 and  $e_2 \notin S$ ,  $u_2$  must be assigned a 0. Similarly, since  $e_3 \in S$ ,  $u_3$  must be assigned a 1 and so on. Finally,  $u_{2k+1} = u_0$  must be assigned a 1. A contradiction. Hence  $u_0$  cannot be assigned a 0.

Again, we claim  $u_0$  cannot be assigned a 1. For if  $u_0$  is assigned a 1, since  $f_1 \notin S$ ,  $v_1$  must be assigned a 0. Again, since  $f_2 \in S$ ,  $v_2$  must be assigned a 1 and so on. Finally,  $v_{2l+1} = u_0$  must be assigned a 0. A contradiction.

Since  $u_0$  can neither be assigned a 0 or a 1, we get a contradiction.

► **Remark.** In the proofs below, we will often encounter similar arguments, where we will have a cycle of dependencies: assigning a particular bit to a location will force the assignment to the next location along the cycle.

**Case: (P2) holds.** (P2) implies that there are edge-disjoint cycles  $C_1$  and  $C_2$  in  $G$  such that,

$$C_1 : u_0 \xrightarrow{e_1} u_1 \xrightarrow{e_2} \cdots \xrightarrow{e_{2k}} u_{2k} = u_0,$$

$$C_2 : v_0 \xrightarrow{f_1} v_1 \xrightarrow{f_2} \cdots \xrightarrow{f_{2l}} v_{2l} = v_0,$$

$2k, 2l \leq n$ , and the third vertices of  $e_1$  and  $f_1$  are the same vertex  $w$ . Let  $S_0 = \{e_1, e_3, \dots, e_{2k-1}\} \cup \{f_2, f_4, \dots, f_{2l}\}$  and  $S_1 = \{e_2, e_4, \dots, e_{2k}\} \cup \{f_1, f_3, \dots, f_{2l-1}\}$ . Note that  $|S_0| = |S_1| \leq n$ . We claim that  $\Phi$  cannot represent any set  $S$  such that

$$S_0 \subseteq S \subseteq \bar{S}_1.$$

In particular,  $\Phi$  cannot represent the set  $S_0$ . Assume  $\Phi$  represents such an  $S$ . Since  $e_1 \in S$ , either the location  $u_0$  or the location  $u_1$  of the memory  $A$  must be assigned a 1, otherwise  $\text{Maj}(A[x(e_1)], A[y(e_1)], A[z(e_1)]) = \text{Maj}(A[u_0], A[u_1], A[w]) = \text{Maj}(0, 0, A[w]) = 0$ . Assume  $u_1$  is assigned a 1. Then, since  $e_2$  is not in the set, using a similar argument,  $u_2$  must be assigned a 0. Similarly,  $u_3$  must be assigned a 1 and so on. Finally,  $u_{2k} = u_0$  must be assigned a 0. Similarly, if  $u_0$  was assigned a 1, then  $u_1$  must be assigned a 0. Thus,  $\text{Maj}(x(e_1), y(e_1), z(e_1)) = \text{Maj}(0, 1, A[w]) = A[w]$ . Hence  $w$  must be assigned a 1.

Again, since  $f_1$  is not in  $S$ , either  $v_0$  or  $v_1$  is assigned a 0. If  $v_1$  is assigned 0,  $v_2$  must be assigned a 1,  $v_3$  a 0, and so on. Finally,  $v_{2l} = v_0$  must be assigned a 1. Similarly, if  $v_0$  is assigned 1, then  $v_1$  is assigned a 0. Therefore,  $\text{Maj}(x(f_1), y(f_1), z(f_1)) = \text{Maj}(A[v_0], A[v_1], A[w]) = \text{Maj}(0, 1, A[w]) = A[w]$ . Since  $f_1 \notin S$ ,  $w$  must be assigned a 0. A contradiction.

**Case: (P3) holds.** (P3) implies that there is a cycle  $C$ :

$$v_0 \xrightarrow{e_1} v_1 \cdots \xrightarrow{e_{2k}} v_{2k} \cdots \xrightarrow{e_{2l}} v_{2l} = v_0,$$

$2k \leq 2l \leq n$  and the third vertices of  $e_1$  and  $e_{2k}$  are the same vertex  $w$ . Let  $S_0 = \{e_1, e_3, \dots, e_{2l-1}\}$  and  $S_1 = \{e_2, e_4, \dots, e_{2k}, \dots, e_{2l}\}$ . Note that  $|S_0| = |S_1| \leq n$ . We claim that  $\Phi$  cannot represent any set  $S$  such that

$$S_0 \subseteq S \subseteq \bar{S}_1.$$

In particular,  $\Phi$  cannot represent the set  $S_0$ . Assume  $\Phi$  represents such an  $S$ . Since  $e_1 \in S$ , either  $v_0$  or  $v_1$  must be assigned a 1. Assume that  $v_1$  is assigned a 1. Then, since  $e_2 \notin S$ ,  $v_2$  must be assigned a 0. Again, since  $e_3 \in S$ ,  $v_3$  must be assigned a 1 and so on. All locations in  $R := \{v_{2r} | 0 \leq r \leq l\}$  must be assigned a 0 and all locations in  $Q := \{v_{2r+1} | 0 \leq r \leq l-1\}$  must be assigned a 1. Else if  $v_0$  is assigned a 1, then all locations in  $R$  must be assigned a 1 and all locations in  $Q$  must be assigned a 0. Now,  $\text{Maj}(x(e_1), y(e_1), z(e_1)) = \text{Maj}(A[v_0], A[v_1], A[w]) = \text{Maj}(0, 1, A[w]) = A[w]$ . Since  $e_1 \in S$ ,  $w$  must be assigned a 0. Similarly,  $\text{Maj}(x(e_{2k}), y(e_{2k}), z(e_{2k})) = \text{Maj}(A[v_{2k-1}], A[v_{2k}], A[w]) = \text{Maj}(0, 1, A[w]) = A[w]$ . Since  $e_{2k} \notin S$ ,  $w$  must be assigned a 1. A contradiction.  $\blacktriangleleft$

In order to prove Lemma 14 we will make use of the following proposition, which is a consequence of a theorem of Alon, Hoory and Linial [2] (see also Ajesh Babu and Radhakrishnan [3]).

**► Proposition 15.** *Fix a graph  $G$  such that the average degree  $d \geq 2$ . Then,*

$$(d-1)^k > |V(G)| \implies \exists \text{ a cycle } C \subseteq E(G), |C| \leq 2k.$$

**Proof of Lemma 14.** Fix an  $(m, n, \left\lfloor \frac{1}{6} m^{1 - \frac{1}{\lfloor \frac{n}{2} \rfloor + 1}} \right\rfloor, 3)$ -scheme  $\Phi$  that uses MAJORITY as the query function. Note that  $s := \left\lfloor \frac{1}{6} m^{1 - \frac{1}{\lfloor \frac{n}{2} \rfloor + 1}} \right\rfloor$  implies

$$m \geq s^{1 + \frac{1}{\lfloor \frac{n}{2} \rfloor}} + 4s + 1. \tag{*}$$

For  $\Phi$  we will come up with a model-graph which is forced, that is, one of (P1), (P2) or (P3) holds. We will start with an initial model-graph  $G$  for  $\Phi$ . We will observe that the average degree of  $G$  is high and invoke Proposition 15 to find a small cycle  $C$ . If  $|C|$  is odd, we will bin it in ODD, delete  $C$  and repeat. If  $|C|$  is even and all the third vertices of the labels of edges in  $C$  are distinct, we will bin  $C$  in EVEN, delete the edges of  $C$  and repeat; otherwise, we will either discover that property (P3) holds or we will modify our model-graph and find an odd cycle in it and bin it in ODD, delete it and repeat. The moment the sum of the lengths of the deleted cycles exceeds  $2s$ , we know either the sum of the lengths of odd or even cycles exceeds  $s$  and two odd cycles intersect or even cycles with distinct third vertices meet, which means either (P1) or (P2) holds. Formally, the procedure can be described as below. We will maintain the following invariant. EVEN will contain edge-disjoint cycles of length even and at most  $n$  each and the third vertices of the labels in such a cycle will be all distinct. ODD will contain edge-disjoint cycles of length odd and at most  $n$ . Furthermore  $([s], E(G) \cup \text{EVEN} \cup \text{ODD})$  will always be a model-graph for  $\Phi$ .

**Step 0: Initialization.**  $\text{EVEN} = \emptyset$ .  $\text{ODD} = \emptyset$ .  $G = ([s], \{x(u) \xleftarrow{u} y(u) | u \in [m]\})$ . Observe  $G$  is a model-graph for  $\Phi$ .

### 38:10 Set Membership with Non-Adaptive Bit Probes

**Step 1.** If  $\sum_{C \in \text{EVEN} \cup \text{ODD}} |C| > 2s$ , END (this ensures that either (P1) or (P2) holds).

Else, using Proposition 15 fix a cycle  $C \subseteq E(G)$  such that  $|C| \leq n$ .

**Step 2.** If  $|C|$  is odd,  $\text{ODD} \leftarrow \text{ODD} \cup \{C\}$  and  $E(G) \leftarrow E(G) \setminus C$  and GOTO Step 1.

**Step 3.** If  $|C|$  is even and all the third vertices of the labels of edges in  $C$  are distinct,

$\text{EVEN} \leftarrow \text{EVEN} \cup \{C\}$  and  $E(G) \leftarrow E(G) \setminus C$  and GOTO Step 1.

**Step 4.** If  $|C|$  is even and the third vertices of the labels of two edges in  $C$  which have an even number of edges between them while traversing the edges of  $C$  in order, then END (Note this means that (P3) holds).

**Step 5.** If  $|C|$  is even and the third vertices of the labels of two edges in  $C$  have an odd number of edges between them (while traversing the edges of  $C$  in order), then represent  $C$  as

$$C : v_0 \xrightarrow{e_1} v_1 \cdots v_{2k} \xrightarrow{e_{2k+1}} v_{2k+1} \cdots v_{2l} \xrightarrow{e_{2l}} v_{2l} = v_0,$$

such that the third vertices of  $e_1$  and  $e_{2k+1}$  are the same vertex  $w$ . We modify the model-graph  $G$  by changing the endpoints of the edges appearing with labels  $e_1, e_{2k+1}$  from  $\{v_0, v_1\}, \{v_{2k}, v_{2k+1}\}$  to  $\{v_1, w\}, \{v_{2k}, w\}$  respectively, thus obtaining a shorter odd cycle  $C'$  in  $G$ :

$$E(G) \leftarrow (E(G) \setminus \{v_0 \xrightarrow{e_1} v_1, v_{2k} \xrightarrow{e_{2k+1}} v_{2k+1}\}) \cup \{v_1 \xrightarrow{e_1} w, v_{2k} \xrightarrow{e_{2k+1}} w\}$$

(Observe:  $G$  with  $E(G) \cup \{e | e \in \text{ODD} \cup \text{EVEN}\}$  continues to be a model-graph for  $\Phi$ ).

$$C' \subseteq E(G) : w \xrightarrow{e_1} v_1 \xrightarrow{e_2} v_2 \cdots v_{2k} \xrightarrow{e_{2k+1}} w$$

is an odd length cycle of length at most  $n$  in  $G$ .

$\text{ODD} \leftarrow \text{ODD} \cup \{C'\}$ .  $E(G) \leftarrow E(G) \setminus C'$ . GOTO Step 1.

In Step 1, if  $|E(G)| \leq 2s$ , then the average degree  $d$  is at least  $\frac{m-2s}{s} > s \lceil \frac{1}{2} \rceil + 2$  (from  $\star$ ) and  $(d-1) \lfloor \frac{n}{2} \rfloor > s$  which implies from Proposition 15 that there is a cycle of length at most  $n$ .

We claim that the procedure terminates only by encountering an END statement in Step 1 or in Step 4. Observe that once the procedure finds a cycle in Step 1, then exactly one of the four if conditions in Steps 2-5 holds. If the procedure does not encounter an END statement in Step 4, then the procedure moves to Step 1 again as each of the Steps 2, 3 and 5 end in a 'GOTO Step 1' statement.

If the procedure encounters the END statement in Step 4, then (P3) holds. If the procedure encounters the END statement in Step 1, then from the pigeonhole principle, either  $\sum_{C \in \text{ODD}} |C| > s$  or  $\sum_{C \in \text{EVEN}} |C| > s$ . In the first case, (P1) holds. In the second case, since each edge in a cycle in EVEN has a distinct third vertex, two cycles in EVEN meet.

Finally, we observe that the procedure terminates. If the procedure does not terminate in Step 4, then the procedure repeatedly finds edge disjoint cycles and deletes them. If the number of edges in the deleted cycle exceeds  $2s$ , then the procedure will terminate when it encounters the END statement in Step 1.  $\blacktriangleleft$

### 3.3 Degree argument

In this section we provide lower bound proofs for the query functions  $(x, y, z) \mapsto (x \wedge y) \oplus z$  and  $(x, y, z) \mapsto 1$  iff  $x + y + z = 1$ .



### 3.3.1 $(x, y, z) \mapsto (x \wedge y) \oplus z$

Let  $\Phi$  be a scheme with  $(x, y, z) \mapsto (x \wedge y) \oplus z$  as the query function. The memory consists of three distinct bit arrays:  $A[1, \dots, s]$ ,  $B[1, \dots, s]$  and  $C[1, \dots, s]$ . For any element  $u \in [m]$ , the scheme probes three locations  $x(u) \in A$ ,  $y(u) \in B$  and  $z(u) \in C$  to determine if  $u$  is in the set or not. We treat each location as a boolean variable. Given any set  $S \subseteq [m]$  of size at most  $n$  the assignment  $\sigma(S) \in \{0, 1\}^{3s}$  to  $A, B$  and  $C$  is such that for all elements  $u \in [m]$ ,  $(x(u) \wedge y(u)) \oplus z(u)$  is 1 iff  $u \in S$ .

We first prove that  $s = \Omega(\sqrt{mn})$  by specializing the lower bound proof in [13] to our case.

► **Definition 16** (Field  $\mathbb{F}_2$ , vector space  $V$ , polynomials  $P_S$ ). Let  $\mathbb{F}_2$  denote the field  $\{0, 1\}$  with mod 2 arithmetic. The query function  $(x, y, z) \mapsto (x \wedge y) \oplus z$  is same as  $(x, y, z) \mapsto xy + z$  (over  $\mathbb{F}_2$ ).

Let  $V$  be the vector space over the field  $\mathbb{F}_2$  of all multilinear polynomials of total degree at most  $2n$  in the  $3s$  variables:  $A[1], \dots, A[s], B[1], \dots, B[s], C[1], \dots, C[s]$  with coefficients coming from  $\mathbb{F}_2$ .

For each set  $S \subseteq [m]$ , we define the polynomial  $P_S$  in  $3s$  variables and coefficients coming from the field  $\mathbb{F}_2$  as follows:

$$P_S = \prod_{u \in S} (x(u)y(u) + z(u)).$$

We make  $P_S$  multilinear by reducing the exponents of each variable using the identity  $x^2 = x$  for each variable  $x$ . This identity holds since we will be considering only 0-1 assignment to the variables.

To prove the theorem for  $(x, y, z) \mapsto (x \wedge y) \oplus z$ , we use the following two lemmas.

► **Lemma 17.** *The set of  $\binom{m}{n}$  multilinear polynomials  $\{P_S : |S| = n\}$  is linearly independent in the vector space  $V$ .*

► **Lemma 18.**  *$V$  has a spanning set of size at most  $\binom{3s+2n}{2n}$ .*

Using these two lemmas, we first prove the theorem and provide the proofs of the lemmas later.

**Proof.** Now, since the size of a linearly independent set is at most the size of a spanning set, using Lemmas 17 and 18, we have

$$\begin{aligned} \binom{m}{n} &\leq \binom{3s+2n}{2n} \\ \implies \left(\frac{m}{n}\right)^n &\leq \left(\frac{e(3s+2n)}{2n}\right)^{2n} \\ \implies 3s &\geq \frac{2}{e}\sqrt{n}(\sqrt{m} - e\sqrt{n}) \\ \implies 3s &\geq \frac{18}{10e}\sqrt{mn} \quad (\text{when } n \leq \frac{m}{900} \{ \implies e\sqrt{n} \leq \frac{1}{10}\sqrt{m} \}). \end{aligned}$$

When  $n \geq \frac{m}{900}$ , the fact that the assignments to the memory for storing different sets of size  $\lceil \frac{m}{900} \rceil$  are different implies that the space required is at least  $\lg \binom{m}{\lceil \frac{m}{900} \rceil} \geq \Omega(m) \geq \Omega(\sqrt{mn})$ . ◀

Now, we prove the two lemmas.

### 38:12 Set Membership with Non-Adaptive Bit Probes

**Proof of Lemma 17.** First observe that any  $S$  of size  $n$ , the polynomial  $P_S$  has  $n$  factors of degree 2 each. Hence, the degree of  $P_S$  is at most  $2n$ .

For sets  $S, S' \subseteq [m]$  of size  $n$  each, the evaluation of the polynomial  $P_S$  on the assignment  $\sigma(S')$  is

$$P_S(\sigma(S')) = \begin{cases} 0 & \text{if } S \neq S' \\ 1 & \text{if } S = S'. \end{cases}$$

Since  $S \neq S'$  and  $|S| = |S'| = n \geq 1$ , there exists  $u \in S$  such that  $u \notin S'$  and thus under the assignment  $\sigma(S')$ , the factor  $x(u)y(u) + z(u)$  in  $P_S(\sigma(S'))$  evaluates to 0. While, when  $S = S'$ , for each  $u \in S$  the factor  $x(u)y(u) + z(u)$  in  $P_S(\sigma(S'))$  evaluates to 1.

In particular, this proves that  $\{P_S : |S| = n\}$  has size  $\binom{m}{n}$ . Further we use this observation below to prove the lemma.

Let  $\sum_{S:|S|=n} \alpha_S P_S = 0$  where each  $\alpha_S \in \mathbb{F}_2$ . To show that the  $P_S$ 's are linearly independent, we need to show that each  $\alpha_S$  is 0. Consider an arbitrary set  $S'$  of size  $n$ , consider the assignment  $\sigma(S')$  to the variables in the above identity.

$$\begin{aligned} 0 &= \sum_{S:|S|=n} \alpha_S P_S(\sigma(S')) \\ &= \alpha_{S'} P_{S'}(\sigma(S')) + \sum_{S:S \neq S', |S|=n} \alpha_S P_S(\sigma(S')) \\ &= \alpha_{S'} P_{S'}(\sigma(S')) \quad (\text{since, } P_S(\sigma(S')) = 0 \text{ for each } S \neq S') \\ &= \alpha_{S'} \quad (\text{since, } P_{S'}(\sigma(S')) = 1). \end{aligned} \quad \blacktriangleleft$$

**Proof of Lemma 18.** The monomials of total degree at most  $2n$  form a spanning set; each polynomial in  $V$  can be written as a linear combination of these monomials. Thus, the size of this spanning set is

$$\sum_{k=0}^{2n} \binom{3s}{k} \leq \binom{3s+2n}{2n},$$

where the last inequality follows from the fact that  $T \mapsto T \cap [3s]$  is an onto map from  $\binom{[3s+2n]}{2n}$  to  $\binom{[3s]}{\leq 2n}$ .  $\blacktriangleleft$

#### 3.3.2 $(x, y, z) \mapsto 1$ iff $x + y + z = 1$

The lower bound proof for  $(x, y, z) \mapsto 1$  iff  $x + y + z = 1$  is similar to the lower bound proof for  $(x, y, z) \mapsto (x \wedge y) \oplus z$ . The only difference here is that instead of looking at the query function over the field  $\mathbb{F}_2$ , we consider the query function over the field  $\mathbb{F}_3$  (the set of three elements  $\{0, 1, 2\}$  with mod 3 arithmetic). Over the field  $\mathbb{F}_3$ , the query function  $(x, y, z) = 1$  iff  $x + y + z = 1$  is same as  $(x, y, z) \mapsto x + y + z + xy + yz + zx$  (a degree 2 polynomial). Accordingly, the multilinear polynomial corresponding to a set  $S$  of size  $n$  is defined to be

$$P_S = \prod_{u \in S} (x(u) + y(u) + z(u) + x(u)y(u) + y(u)z(u) + z(u)x(u)).$$

where we reduce the exponents using the identity  $x^2 = x$  for each variable  $x$  (this identity holds as we consider only 0-1 assignments). Notice that  $P_S$  has degree at most  $2n$  and the rest of the proof is same as before.

The proofs for functions from the remaining classes, the proofs of Theorems 4(c) and 3 are available in the full version of the paper [6].

## References

- 1 Noga Alon and Uriel Feige. On the power of two, three and four probes. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 346–354, 2009. URL: <http://dl.acm.org/citation.cfm?id=1496770.1496809>.
- 2 Noga Alon, Shlomo Hoory, and Nathan Linial. The Moore bound for irregular graphs. *Graphs and Combinatorics*, 18(1):53–57, 2002. doi:10.1007/s003730200002.
- 3 Ajesh Babu and Jaikumar Radhakrishnan. An entropy based proof of the Moore bound for irregular graphs. *CoRR*, abs/1011.1058, 2010. URL: <http://arxiv.org/abs/1011.1058>.
- 4 Harry Buhrman, Peter Bro Miltersen, Jaikumar Radhakrishnan, and Srinivasan Venkatesh. Are bitvectors optimal? *SIAM J. Comput.*, 31(6):1723–1744, 2002. doi:10.1137/S0097539702405292.
- 5 Mohit Garg and Jaikumar Radhakrishnan. Set membership with a few bit probes. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 776–784, 2015. doi:10.1137/1.9781611973730.53.
- 6 Mohit Garg and Jaikumar Radhakrishnan. Set membership with non-adaptive bit probes. *CoRR*, abs/1612.09388, 2016. URL: <http://arxiv.org/abs/1612.09388>.
- 7 Michael T. Goodrich and Michael Mitzenmacher. Invertible Bloom lookup tables. In *49th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2011, Allerton Park & Retreat Center, Monticello, IL, USA, 28-30 September, 2011*, pages 792–799, 2011. doi:10.1109/Allerton.2011.6120248.
- 8 Moshe Lewenstein, J. Ian Munro, Patrick K. Nicholson, and Venkatesh Raman. Improved explicit data structures in the bitprobe model. In *Algorithms – ESA 2014 – 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 630–641, 2014. doi:10.1007/978-3-662-44777-2\_52.
- 9 Michael Luby, Michael Mitzenmacher, Mohammad Amin Shokrollahi, and Daniel A. Spielman. Efficient erasure correcting codes. *IEEE Trans. Information Theory*, 47(2):569–584, 2001. doi:10.1109/18.910575.
- 10 Ali Makhdoumi, Shao-Lun Huang, Muriel Médard, and Yury Polyanskiy. On locally decodable source coding. In *2015 IEEE International Conference on Communications, ICC 2015, London, United Kingdom, June 8-12, 2015*, pages 4394–4399, 2015. doi:10.1109/ICC.2015.7249014.
- 11 Marvin Minsky and Seymour Papert. *Perceptrons*. MIT press, Cambridge, MA, 1969.
- 12 Jaikumar Radhakrishnan, Venkatesh Raman, and S. Srinivasa Rao. Explicit deterministic constructions for membership in the bitprobe model. In *Algorithms – ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, pages 290–299, 2001. doi:10.1007/3-540-44676-1\_24.
- 13 Jaikumar Radhakrishnan, Pranab Sen, and Srinivasan Venkatesh. The quantum complexity of set membership. *Algorithmica*, 34(4):462–479, 2002. doi:10.1007/s00453-002-0979-0.
- 14 Jaikumar Radhakrishnan, Smit Shah, and Saswata Shannigrahi. Data structures for storing small sets in the bitprobe model. In Mark de Berg and Ulrich Meyer, editors, *Algorithms – ESA 2010, 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part II*, volume 6347 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 2010. doi:10.1007/978-3-642-15781-3\_14.
- 15 Emanuele Viola. Bit-probe lower bounds for succinct data structures. *SIAM J. Comput.*, 41(6):1593–1604, 2012. doi:10.1137/090766619.
- 16 Wikiversity. The 22 becs, 3-ary boolean functions – wikiversity, 2016. [Online; accessed 7-August-2016]. URL: [https://en.wikiversity.org/w/index.php?title=3-ary\\_Boolean\\_functions&oldid=1587287](https://en.wikiversity.org/w/index.php?title=3-ary_Boolean_functions&oldid=1587287).



# Pro-Aperiodic Monoids via Saturated Models\*

Samuel J. v. Gool<sup>1</sup> and Benjamin Steinberg<sup>2</sup>

- 1 Institute for Logic, Language and Computation, University of Amsterdam, Amsterdam, The Netherlands; and  
Department of Mathematics, City College of New York, New York, NY, USA  
samvangool@me.com
- 2 Department of Mathematics, City College of New York, New York, NY, USA  
bsteinberg@ccny.cuny.edu

---

## Abstract

We apply Stone duality and model theory to study the structure theory of free pro-aperiodic monoids. Stone duality implies that elements of the free pro-aperiodic monoid may be viewed as elementary equivalence classes of pseudofinite words. Model theory provides us with saturated words in each such class, i.e., words in which all possible factorizations are realized. We give several applications of this new approach, including a solution to the word problem for  $\omega$ -terms that avoids using McCammond's normal forms, as well as new proofs and extensions of other structural results concerning free pro-aperiodic monoids.

**1998 ACM Subject Classification** F.4.1 Model Theory, F.4.3 Algebraic Language Theory

**Keywords and phrases** aperiodic monoids, profinite monoids, Stone duality, saturated models

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.39

## 1 Introduction

The pseudovariety of aperiodic monoids has long played a fundamental role in finite semigroup theory and automata theory. The famous Schützenberger theorem [32] proved that the aperiodic monoids recognize precisely the star-free languages. This class, which also coincides with the class of languages recognizable by counter-free automata, was later shown by McNaughton and Papert [27] to be the class of first-order definable languages, also see Straubing's book [35]. Algorithmic questions about aperiodic languages lead to challenges that the algebraic approach can often help resolve. In particular, aperiodic monoids play a prime role in two of the oldest and most difficult open problems in automata theory: the dot-depth problem [10, 29, 28] and the Krohn-Rhodes complexity problem [23].

Within this algebraic approach to aperiodic languages, *free pro-aperiodic monoids* are a useful tool. The importance of profinite monoids in automata theory and finite semigroup theory was first highlighted, starting in the late eighties, by Almeida [1]; also see the more recent monograph by Rhodes and the second-named author [30], or Straubing and Weil's handbook article [36]. The structure of free pro-aperiodic monoids has been studied recently by several authors, e.g., [4, 20, 22, 6], but many difficult questions remain open. Many of the existing results about free pro-aperiodic monoids are about the submonoid of elements definable by  $\omega$ -terms and rely on an ingenious normal form algorithm due to

---

\* The first-named author was supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant #655941; the second-named author was supported by Simons Foundation #245268, United States – Israel Binational Science Foundation #2012080 and NSA MSP #H98230-16-1-0047.



McCammond [25], which solves the word problem for  $\omega$ -terms. In our new approach to pro-aperiodic monoids we replace the use of normal forms by the model-theoretic concept of saturated word.

### Contributions of this paper

Our contributions can be separated into useful theoretical results about pro-aperiodic monoids on the one hand, and applications of this theory to structural and decidability results for pro-aperiodic monoids on the other hand.

The main theoretical results (Section 4) are the following:

- Identification of the free pro-aperiodic monoid with a monoid of elementary equivalence classes of pseudofinite words (Theorem 10), and of homomorphisms with substitutions (Theorem 12).
- Substitution-invariance of  $\omega$ -saturated words (Theorem 15).
- A classification of the factors of the image of a substitution (Theorem 18).

The main applications of these results (Sections 5–7) are the following:

- Decidability of the word problem for aperiodic  $\omega$ -terms that avoids McCammond’s normal forms (Section 5).
- New, easier proofs of known facts about factors of  $\omega$ -terms, notably that factors of  $\omega$ -terms are given by  $\omega$ -terms (Theorem 22).
- Having a well-quasi-order of factors, prefixes, or suffixes is stable under substitutions (Theorem 23); in particular,  $\omega$ -terms have these properties (Corollary 26).
- Having a regular language of factors, prefixes, or suffixes is stable under non-erasing substitutions (Theorem 28); in particular,  $\omega$ -terms have these properties (Corollary 29).

### Methodology

Early on in the study of profinite monoids, Almeida [1] made the important observation that the Boolean algebra of clopen subsets of the free pro- $\mathbf{V}$  monoid can be identified with the Boolean algebra of  $\mathbf{V}$ -recognizable languages; in other words, the free pro- $\mathbf{V}$  monoid is the *Stone dual* of the Boolean algebra of  $\mathbf{V}$ -recognizable languages. In recent years, a number of authors have made explicit use of Stone duality theory to redevelop and expand the foundations of the profinite approach to studying varieties of languages in the sense of Eilenberg [14]; most closely related to our work here is the work of Gehrke, Pin et al. [16, 17, 18, 15], Bojańczyk [9], and Rhodes and the second-named author [30, Chapter 8].

In this paper we apply Stone duality in the case where  $\mathbf{V}$  is the variety of aperiodic monoids to study the structure theory of free pro-aperiodic monoids. The crucial idea is that we can view elements of the free pro-aperiodic monoid as complete first-order theories containing the theory of finite words. By the completeness theorem of first-order logic, these in turn can be identified with elementary equivalence classes of so-called pseudofinite words, models of the theory of finite words. These models can be concatenated in a natural way, allowing us to recover the algebraic structure as well as the topological structure from this approach. See Section 3 for details.

In addition to Stone duality, the correspondence between aperiodic monoids and first-order logic allows us to import techniques from *first-order model theory*, in particular *Ehrenfeucht-Fraïssé games*, which are commonly used in logic on words, and more importantly  *$\omega$ -saturated models*. In our context, an  $\omega$ -saturated model is a word where every possible factorization is realized in the word itself; see Section 4 for details. These  $\omega$ -saturated words allow us to perform an explicit combinatorial analysis of the factors of elements of pro-aperiodic monoids.

We leave model-theoretic prerequisites to a minimum; for the precise correspondence to the general model-theoretic concepts and other technical proofs, we refer the reader to our extended technical report [19], where we provide more background information than space permits here.

## 2 Logic on words

We briefly recall the necessary preliminaries on logic on words. We mostly follow the standard terminology as in, e.g., [35].

**Words.** In what follows,  $A, B, \dots$  denote non-empty finite alphabets. In this paper, by a *word* over  $A$ , or *A-word*, we mean a tuple  $W = (|W|, <^W, (P_a^W)_{a \in A})$ , where

- $|W|$  is a set,
- the relation  $<^W$  is a discrete linear order with endpoints on  $|W|$ , i.e., there are a first and a last element, every element except the last has a unique immediate  $<^W$ -successor and every element except the first has a unique immediate  $<^W$ -predecessor,
- $(P_a^W)_{a \in A}$  is a partition of  $|W|$ .

An *A-word* is called *finite* if  $|W|$  is finite. We denote by  $\varepsilon$  the unique *A-word* with  $|W| = \emptyset$ .

Let  $W$  be an *A-word*. For any  $i \in |W|$ , we write  $W(i)$  for the unique letter  $a$  such that  $i \in P_a^W$ ; we write  $W(<i)$  for the *A-word* obtained by restricting  $W$  to the set of positions strictly less than  $i$  (the *ray* left of  $i$ ), and similarly we define  $W(>i)$  (the *ray* right of  $i$ ). For  $i, j \in |W|$ , we write  $W(i, j)$  for the *open interval*,  $W[i, j]$  for the *closed interval* of  $W$  between these positions, and *half-open intervals*  $W[i, j)$  and  $W(i, j]$ . Since the order on an *A-word* is discrete with endpoints, any (half-)open interval or ray can be written as a closed interval; we use this fact without mention in what follows.

**Logic.** An *atomic formula* is an expression of the form  $x < y$  or  $P_a(x)$ , where, here and in what follows,  $x$  and  $y$  denote first-order variables and  $a$  denotes a letter from the alphabet. A *first-order formula* is an expression built up from atomic formulas by inductively applying the connectives  $\wedge, \vee, \neg, \rightarrow, \exists x$ , and  $\forall x$ . The notation  $\varphi(\bar{x})$  indicates that all the variables that occur freely in  $\varphi$  lie in  $\bar{x}$ . A *sentence* is a formula without free occurrences of variables.

For a first-order formula  $\varphi(\bar{x})$ , a word  $W$  and an assignment  $\bar{x}^W$  of the variables  $\bar{x}$  in  $|W|$ , one defines  $W, \bar{x}^W \models \varphi(\bar{x})$  if  $\varphi(\bar{x})$  is true in  $W$  under the assignment  $\bar{x}^W$ . The *language defined by an FO-sentence*  $\varphi$  is the set  $L_\varphi$  of *finite A-words*  $W$  such that  $W \models \varphi$ .

The first-order definable languages form a strict subclass of the regular languages. Indeed, recall [32, 27] that *a language  $L$  of finite A-words is first-order definable if and only if the syntactic monoid of  $L$  is finite and aperiodic*. This fundamental result is the starting point for our perspective on the free pro-aperiodic monoid, cf. Section 4 below.

**Quantifier depth and games.** The *quantifier depth* of a formula  $\varphi$  is the maximum nesting depth of quantifiers in  $\varphi$ . If  $U$  and  $V$  are *A-words*, we write  $U \equiv_k V$  if  $U$  and  $V$  satisfy exactly the same first-order sentences  $\varphi$  of quantifier depth at most  $k$ ; in this case, we say that  $U$  and  $V$  are *elementarily equivalent up to quantifier depth  $k$*  or simply  *$k$ -equivalent*. We write  $U \equiv V$  if  $U \equiv_k V$  for all  $k$ , i.e.,  $U$  and  $V$  satisfy exactly the same first-order sentences; in this case, we say that  $U$  and  $V$  are *elementarily equivalent*. Importantly, for each  $k \geq 0$ , there are only *finitely* many  $k$ -equivalence classes, and each such class is definable by a first-order sentence [21, Thm. 3.3.2], see also [35, Sec. IV.1].

One can alternatively describe  $k$ -equivalence using Ehrenfeucht-Fraïssé games, cf., e.g., [21, Ch. 3] or [35, Sec. IV.1]. These games allow us to prove the following lemma. If  $U$  and  $V$  are  $A$ -words,  $i \in |U|$  and  $j \in |V|$ , we say that the position  $i$  in  $U$   $k$ -corresponds to the position  $j$  in  $V$  provided that  $U(<i) \equiv_k V(<j)$ ,  $U(i) = V(j)$ , and  $U(>i) \equiv_k V(>j)$ .

► **Lemma 1.** *Let  $U$  and  $V$  be  $A$ -words. For all  $k \geq 0$ ,  $U \equiv_{k+1} V$  if and only if for every  $i \in |U|$ , there exists  $j \in |V|$  that  $k$ -corresponds to  $i$ , and for every  $j \in |V|$ , there exists  $i \in |U|$  that  $k$ -corresponds to  $j$ .*

### 3 Pro-aperiodic monoids as Stone dual spaces

In this section, we define free pro-aperiodic monoids and a new object  $\Lambda(A)$ , and identify them with spaces of elementary equivalence classes of models.

**Pro-aperiodic monoids.** A *profinite monoid* is an inverse limit of finite discrete monoids in the category of topological monoids (i.e., monoids whose underlying set is equipped with a topology in which the monoid operation is continuous). Equivalently, a profinite monoid is a topological monoid whose underlying space is a *Boolean* or *Stone space*, i.e., compact, Hausdorff and zero-dimensional. Profinite monoids inherit many properties of finite monoids. Of particular interest to us is the fact that any element  $x$  in a profinite monoid has a unique idempotent, denoted  $x^\omega$ , in its orbit-closure  $\overline{\{x^n \mid n \geq 1\}}$ . A *pro-aperiodic* monoid  $M$  is a profinite monoid in which  $x^\omega = x^\omega x$  for all  $x \in M$ . Equivalently, a pro-aperiodic monoid is an inverse limit of finite aperiodic monoids; here, finite monoids are equipped with the discrete topology, and the inverse limit is taken in the category of topological monoids.

The *free pro-aperiodic monoid* generated by a finite set  $A$  is a pro-aperiodic monoid  $\widehat{F}_A(A)$  containing  $A$  such that any function  $f: A \rightarrow M$ , with  $M$  a finite aperiodic monoid, extends uniquely to a continuous homomorphism  $\bar{f}: \widehat{F}_A(A) \rightarrow M$ , where  $M$  is given the discrete topology. The free pro-aperiodic monoid is unique up to topological isomorphism, and the same extension property still holds if in the previous sentence  $M$  is replaced by an arbitrary pro-aperiodic monoid.

Let  $M$  be a monoid and  $u, v$  elements of  $M$ . Then  $u \leq_{\mathcal{J}} v$  means that there exist  $x$  and  $y$  such that  $u = xvy$ , and in this case  $v$  is called a *factor* of  $u$ ;  $u \leq_{\mathcal{L}} v$  means that there exists  $x$  such that  $u = xv$ , and in this case  $v$  is called a *suffix* of  $u$ ;  $u \leq_{\mathcal{R}} v$  means that there exists  $y$  such that  $u = vy$ , and in this case  $v$  is called a *prefix* of  $u$ . Each of these relations is a quasi-order on  $M$ ; the equivalence relations they induce are denoted  $\mathcal{J}$ ,  $\mathcal{L}$  and  $\mathcal{R}$ , e.g.,  $u \mathcal{J} v$  means that  $u \leq_{\mathcal{J}} v$  and  $v \leq_{\mathcal{J}} u$ .

**Stone duality.** We make use of *Stone duality*, which, we briefly recall, is the dual equivalence between the categories of Boolean algebras and Boolean spaces that takes a Boolean space  $X$  to its algebra  $\mathcal{K}(X)$  of clopen sets, and a Boolean algebra  $B$  to the set of ultrafilters  $\text{Spec}(B)$  of  $B$ , which is given a Boolean topology by declaring, for each  $L \in B$ , the set  $\widehat{L} := \{x \in \text{Spec}(B) \mid L \in x\}$  to be open. Stone's duality theorem [34] says that the assignment  $L \mapsto \widehat{L}$  is an isomorphism from  $B$  to  $\mathcal{K}(\text{Spec}(B))$ , and that moreover Boolean algebra homomorphisms  $B_1 \rightarrow B_2$  are in natural bijection with continuous functions  $\text{Spec}(B_2) \rightarrow \text{Spec}(B_1)$ .

**Spaces of first-order theories.** Recall that, for  $T$  a set of sentences in first-order logic, the *Lindenbaum-Tarski algebra*  $\text{LT}(T)$  of  $T$  is the Boolean algebra of  $T$ -equivalence classes of first-order sentences. Here, two first-order sentences  $\varphi$  and  $\psi$  are  $T$ -equivalent if, in any



model  $W$  where all sentences in  $T$  are true,  $\varphi$  and  $\psi$  are either both true or both false. Stone dual spaces of Lindenbaum-Tarski algebras are well understood in logic.

► **Proposition 2.** *Let  $T$  be a set of first-order sentences. The Stone dual space of  $\text{LT}(T)$  is homeomorphic to the Boolean space  $X$  whose points are elementary equivalence classes of models of  $T$ , in which the clopen sets are exactly the truth sets*

$\hat{\varphi} := \{x \in X \mid \varphi \text{ is true in the models in the class } x\}$ , for  $\varphi$  any first-order sentence.

**Theories of  $A$ -words.** For a finite alphabet  $A$ , we denote by  $T_A$  the (finitely axiomatized) *theory of  $A$ -words*, that is, the set of first-order sentences deducible from axioms expressing that the order is a discrete linear order with endpoints, and that exactly one letter predicate holds at each position. We further let  $T_A^{\text{fin}}$  denote the *theory of finite  $A$ -words*, i.e., the set of first-order sentences that are true in all finite  $A$ -words. A model of the theory  $T_A^{\text{fin}}$  is called a *pseudofinite  $A$ -word*.<sup>1</sup> The theories  $T_A$  and  $T_A^{\text{fin}}$  do not coincide in general; in fact, they coincide only if the alphabet  $A$  contains a single letter. In this case, both theories  $T_A$  and  $T_A^{\text{fin}}$  are the theory of discrete linear orders with endpoints, with a unary predicate that is true everywhere.

As soon as  $A$  contains at least two letters, the situation is very different. In particular, there are  $A$ -words that are not pseudofinite.

► **Example 3.** Let  $W$  be the word  $aaaa \dots \dots bbbb$ , i.e.,  $W$  is the word over the alphabet  $\{a, b\}$  with underlying order  $\mathbb{N} + \mathbb{N}^{\text{op}}$ , where  $W(i) = a$  for all  $i \in \mathbb{N}$  and  $W(i) = b$  for all  $i \in \mathbb{N}^{\text{op}}$ . The sentence  $\exists x P_a(x) \rightarrow \exists x (P_a(x) \wedge \forall y (y > x \rightarrow \neg P_a(y)))$ , expressing ‘if there exists an  $a$ -position, then there exists a last such’ is true in every finite  $A$ -word, and therefore lies in  $T_A^{\text{fin}}$ , but it fails to hold in  $W$ . Thus,  $W$  is not pseudofinite.

► **Remark.** In [19, Sec. 4], we discuss the axiomatizability of the theory  $T_A^{\text{fin}}$  of finite  $A$ -words. In particular, we show that the theory is not finitely axiomatizable, and we give an axiomatization of it using an axiom scheme similar to the one given by Doets [13].

**Pro-aperiodic monoids as spaces of theories.** We will denote by  $\Lambda(A)$  the Stone dual space of  $\text{LT}(T_A)$ . By Proposition 2,  $\Lambda(A)$  is the space of elementary equivalence classes of  $A$ -words. Combining this with the characterization theorem [32, 27] that first-order languages are exactly the aperiodic-recognizable languages, we obtain the following lemma and proposition.

► **Lemma 4.** *The Lindenbaum-Tarski algebra  $\text{LT}(T_A^{\text{fin}})$  of the theory  $T_A^{\text{fin}}$  is isomorphic to the algebra  $\text{Rec}_A(A)$  of aperiodic-recognizable languages of finite  $A$ -words.*

► **Proposition 5.** *The subspace of  $\Lambda(A)$  consisting of the elementary equivalence classes of pseudofinite  $A$ -words is homeomorphic to the space underlying the free pro-aperiodic monoid over  $A$ .*

We will see in the next section that there is a natural multiplication on  $\Lambda(A)$  which makes it into a pro-aperiodic monoid, and makes the homeomorphism of Proposition 5 into a topological isomorphism of pro-aperiodic monoids.<sup>2</sup>

<sup>1</sup> This is an instance of the general model-theoretic use of the term ‘pseudofinite’, cf., e.g., [37].

<sup>2</sup> Here and in what follows, we use the term ‘homeomorphism’ to indicate an isomorphism in the category of topological spaces, and ‘topological isomorphism’ for an isomorphism in the category of topological monoids.

## 4 Substitutions, saturated words, and factorizations

This is the main theoretical section of the paper. We introduce substitutions and saturated words and state our main results about them.

### Substitutions

Suppose that  $V$  is a word over a finite alphabet  $B$ , and that for each  $b \in B$ ,  $U_b$  is a word over a finite alphabet  $A$ . The *substitution* of the  $A$ -words  $(U_b)_{b \in B}$  into the  $B$ -word  $V$  is the  $A$ -word  $W = V[b/U_b]$  defined as follows.

- The underlying order of  $W$  is the *lexicographic order* on the disjoint union  $|W| := \bigsqcup_{i \in |V|} |U_{V(i)}|$ , i.e.,  $(i, j) <^W (i', j') \stackrel{\text{def}}{\iff} i <^V i'$ , or  $i = i'$  and  $j <^{U_{V(i)}} j'$ .
- The letter at position  $(i, j)$  in  $W$  is the letter at position  $j$  in  $U_{V(i)}$ .

There are two important special cases of substitution. If  $U_0$  and  $U_1$  are  $A$ -words, then the *concatenation*  $U_0 \cdot U_1$  of  $U_0$  and  $U_1$  is defined as the substitution of  $(U_b)_{b \in \{0,1\}}$  into the  $\{0,1\}$ -word  $01$ . If  $U$  is an  $A$ -word and  $\lambda$  is a discrete linear order with endpoints, then the  $\lambda$ -*power*  $U^\lambda$  of  $U$  is defined as the substitution of  $U_b = U$  into the unique  $\{b\}$ -word with underlying order  $\lambda$ .

Importantly, the operation of substitution respects the equivalence relations  $\equiv_k$ .

► **Proposition 6.** *Let  $k \geq 0$ . For any finite alphabets  $A$  and  $B$  and any  $B$ -indexed collections of  $A$ -words  $(U_b)_{b \in B}$  and  $(U'_b)_{b \in B}$  such that  $U_b \equiv_k U'_b$  for each  $b \in B$ , if  $V$  and  $V'$  are  $A$ -words such that  $V \equiv_k V'$ , then  $V[b/U_b] \equiv_k V'[b/U'_b]$ .*

► **Corollary 7.** *If  $V \equiv V'$  and  $U_b \equiv U'_b$  for each  $b \in B$ , then  $V[b/U_b] \equiv V'[b/U'_b]$ .*

### Pro-aperiodic monoids of elementary equivalence classes

Corollary 7 in particular implies that there is a well-defined binary operation of concatenation on the set  $\Lambda(A)$  of elementary equivalence classes of  $A$ -words.

► **Theorem 8.** *The Stone space  $\Lambda(A)$ , equipped with the operation of concatenation up to elementary equivalence, is a pro-aperiodic monoid. The  $\omega$ -power of an element  $[U]_{\equiv}$  is  $[U^\lambda]_{\equiv}$ , where  $\lambda$  is any infinite discrete linear order with endpoints.*

We showed in Proposition 5 that the space underlying the free pro-aperiodic monoid,  $\widehat{F}_{\mathbf{A}}(A)$ , is homeomorphic to the subspace of  $\Lambda(A)$  consisting of the elementary equivalence classes of pseudofinite  $A$ -words. Indeed, we can prove more.

► **Proposition 9.** *The set of elementary equivalence classes of pseudofinite  $A$ -words is a topologically closed submonoid of  $\Lambda(A)$  which is closed under taking factors.*

We now establish our first main theoretical contribution.

► **Theorem 10.** *Let  $A$  be a finite alphabet. The free pro-aperiodic monoid over  $A$  is topologically isomorphic to the pro-aperiodic monoid of elementary equivalence classes of pseudofinite  $A$ -words.*

► **Convention.** In view of Theorem 10, we henceforth identify the free pro-aperiodic monoid  $\widehat{F}_{\mathbf{A}}(A)$  with the closed submonoid  $\text{PF}(A)$  of  $\Lambda(A)$  consisting of the elementary equivalence classes of pseudofinite  $A$ -words.

► **Remark.** The pro-aperiodic monoids  $\Lambda(A)$  and  $\widehat{F}_{\mathbf{A}}(A)$  can also be usefully described as inverse limits of chains of finite monoids of  $k$ -equivalence classes, cf. [19, Sec. 3].

► **Proposition 11.** *Let  $(U_b)_{b \in B}$  be a  $B$ -indexed collection of  $A$ -words. The function  $f: \Lambda(B) \rightarrow \Lambda(A)$ , which sends an element  $[V]_{\equiv}$  of  $\Lambda(B)$  to  $[V[b/U_b]]_{\equiv}$ , is a well-defined continuous homomorphism.*

We call a continuous homomorphism  $f: \Lambda(B) \rightarrow \Lambda(A)$  a *substitution* if it arises as in Proposition 11. Restricting attention to the free pro-aperiodic monoids, we obtain the following theorem.

► **Theorem 12.** *The continuous homomorphisms from  $\widehat{F}_{\mathbf{A}}(B)$  to  $\widehat{F}_{\mathbf{A}}(A)$  are exactly the substitutions of pseudofinite  $A$ -words into pseudofinite  $B$ -words.*

### Saturated words

Let  $A$  be a finite alphabet and let  $U$  be an  $A$ -word. For any position  $i \in |U|$ , define the triple  $t^U(i) := ([U(<i)]_{\equiv}, U(i), [U(>i)]_{\equiv})$ , an element of the Cartesian product  $\Lambda(A) \times A \times \Lambda(A)$ . We call  $t^U(i)$  the (complete 1-)type of  $i$  in  $U$  (see [19, Prop. A.3] for the correspondence with types in model theory). We refer to the product space  $\Lambda(A) \times A \times \Lambda(A)$ , where the middle component  $A$  has the discrete topology, as the *type space*.

We write  $\text{RT}(U)$  for the *set of types realized in  $U$* , i.e.,  $\text{RT}(U)$  is the subset  $\{t^U(i) \mid i \in |U|\}$  of the type space. If  $V$  is an  $A$ -word elementarily equivalent to  $U$  and  $j \in |V|$ , then we say the type  $t^V(j)$  is *consistent with  $U$* . We write  $\text{CT}(U)$  for the *set of types consistent with  $U$* , i.e.,  $\text{CT}(U)$  is the subset  $\{t^V(j) \mid V \equiv U, j \in |V|\}$  of the type space.

► **Definition 13.** Let  $U$  be an  $A$ -word. We say that  $U$  is

- *weakly saturated* if every type consistent with  $U$  is realized in  $U$ , i.e.,  $\text{CT}(U) = \text{RT}(U)$ ;
- *$\omega$ -saturated* if every closed interval  $U[i, j]$  in  $U$  is weakly saturated;
- *countably saturated* if the underlying set  $|U|$  is countable and  $U$  is  $\omega$ -saturated.

It is well-known in model theory that any elementary equivalence class contains an  $\omega$ -saturated model, which typically has an uncountable underlying set; also see [19, Prop. A.5].

► **Example 14.** Consider the case of a one-letter alphabet,  $\{a\}$ . All infinite  $\{a\}$ -words are elementarily equivalent and pseudofinite. Hence,  $\widehat{F}_{\mathbf{A}}(\{a\}) = \Lambda(\{a\})$  is topologically isomorphic to the topological monoid  $\mathbb{N} \cup \{\omega\}$ , i.e., the one-point compactification of  $\mathbb{N}$  with the usual addition, where  $\omega$  is an absorbing element. The space of types of  $\{a\}$ -words is  $\widehat{F}_{\mathbf{A}}(\{a\}) \times \{a\} \times \widehat{F}_{\mathbf{A}}(\{a\})$ . Concretely, types of  $\{a\}$ -words are of the following four forms:

- $(a^n, a, a^m)$  for  $n, m \in \mathbb{N}$ ;
- $(a^n, a, a^\omega)$  for  $n \in \mathbb{N}$ ;
- $(a^\omega, a, a^m)$  for  $m \in \mathbb{N}$ ;
- $(a^\omega, a, a^\omega)$ .

Consider the following infinite  $\{a\}$ -words:

1.  $W_1 := a^{\mathbb{N} + \mathbb{N}^{\text{op}}}$ ,
2.  $W_2 := a^{\mathbb{N} + \mathbb{Z} + \mathbb{N}^{\text{op}}}$ ,
3.  $W_3 := a^{\mathbb{N} + \mathbb{Q} \times \mathbb{Z} + \mathbb{N}^{\text{op}}}$ .

The word  $W_1$  is not weakly saturated, because the elementarily equivalent word  $W_2$  realizes the type  $(a^\omega, a, a^\omega)$ , which is not realized in  $W_1$ , that is,  $(a^\omega, a, a^\omega) \in \text{CT}(W_1) \setminus \text{RT}(W_1)$ . The word  $W_2$  is weakly saturated, because it realizes all the types. However,  $W_2$  is not  $\omega$ -saturated, because the ray to the left of  $i$ , where  $i$  is any point in the summand  $\mathbb{Z}$ , is isomorphic to  $W_1$ , and not weakly saturated. Notice that any closed interval in the word in  $W_3$  is either finite or isomorphic to  $W_3$ , using the well-known fact that any open interval in the order  $\mathbb{Q}$  is isomorphic to  $\mathbb{Q}$  (cf. e.g., [21, p. 100]). Since finite words and  $W_3$  are weakly

saturated, the word  $W_3$  is in fact  $\omega$ -saturated. Since  $|W_3|$  is countable,  $W_3$  is countably saturated.

The key technical result about saturated words is the following.

► **Theorem 15.** *If  $V$  is an  $\omega$ -saturated  $B$ -word and  $(U_b)_{b \in B}$  is a  $B$ -indexed collection of  $A$ -words that are  $\omega$ -saturated, then  $V[b/U_b]$  is  $\omega$ -saturated.*

Note that Theorem 15 has the following immediate consequence.

► **Corollary 16.** *The concatenation of two  $\omega$ -saturated words is  $\omega$ -saturated. The power of an  $\omega$ -saturated word by an  $\omega$ -saturated linear order is  $\omega$ -saturated.*

### Factorizations

A useful fact about  $\omega$ -saturated words is that they realize any finite factorization of their elementary equivalence class.

► **Lemma 17.** *Let  $W$  be an  $\omega$ -saturated  $A$ -word and suppose that  $W \equiv W_1 \cdots W_n$  with  $W_1, \dots, W_n$  non-empty  $A$ -words. Then we can find positions  $i_1 < i_2 < \cdots < i_{n-1}$  in  $|W|$  such that  $W(<i_1) \equiv W_1$ ,  $W(\geq i_{n-1}) \equiv W_n$  and  $W[i_j, i_{j+1}) \equiv W_{j+1}$  for  $1 \leq j \leq n-2$ .*

As a first application of Theorem 15, we will now analyze what factors of the result of a substitution can look like in  $\Lambda(A)$  (and hence, by Proposition 9, in  $\widehat{F}_A(A)$ ).

► **Theorem 18.** *Let  $f: \Lambda(B) \rightarrow \Lambda(A)$  be a substitution. Let  $v \in \Lambda(B)$  and  $w \in \Lambda(A)$ .*

1.  *$f(v) \leq_{\mathcal{J}} w$  if and only if one of the following holds:*
  - (a)  *$w = \varepsilon$ , or*
  - (b) *there exists  $b \in B$  such that  $v \leq_{\mathcal{J}} b$  and  $f(b) \leq_{\mathcal{J}} w$ , or*
  - (c) *there exist  $b_1, b_2 \in B$ ,  $x, y \in \Lambda(A)$  and  $z \in \Lambda(B)$  such that  $v \leq_{\mathcal{J}} b_1 z b_2$ ,  $f(b_1) \leq_{\mathcal{L}} x$ ,  $f(b_2) \leq_{\mathcal{R}} y$ , and  $w = x f(z) y$ .*
2.  *$f(v) \leq_{\mathcal{R}} w$  if and only if  $w = \varepsilon$  or there exist  $x \in \Lambda(A)$ ,  $b \in B$ , and  $z \in \Lambda(B)$  such that  $f(b) \leq_{\mathcal{R}} x$ ,  $v \leq_{\mathcal{R}} z b$ , and  $w = f(z) x$ .*
3.  *$f(v) \leq_{\mathcal{L}} w$  if and only if  $w = \varepsilon$  or there exist  $y \in \Lambda(A)$ ,  $c \in B$ , and  $z \in \Lambda(B)$  such that  $f(c) \leq_{\mathcal{L}} y$ ,  $v \leq_{\mathcal{L}} c z$ , and  $w = y f(z)$ .*

Theorem 18 can be viewed as extension of [7, Lemma 8.2], where the special case that  $v$  is a finite word is handled in the context of the free profinite monoid.

We may usefully summarize Theorem 18 by recalling some more notation. For any  $u \in \Lambda(A)$ , write  $\uparrow_{\mathcal{J}} u := \{w \in \Lambda(A) \mid w \geq_{\mathcal{J}} u\}$  for the set of factors of  $u$ , and similarly  $\uparrow_{\mathcal{R}} u$  for the set of prefixes of  $u$ , and  $\uparrow_{\mathcal{L}} u$  for the set of suffixes of  $u$ . Also, for any subset  $L$  of  $\Lambda(A)$  and  $a, b \in A$ , write  $a^{-1}L := \{u \in \Lambda(A) \mid au \in L\}$  and  $Lb^{-1} := \{u \in \Lambda(A) \mid ub \in L\}$ , and  $a^{-1}Lb^{-1} := \{u \in \Lambda(A) \mid aub \in L\}$ . Finally, if  $u \in \Lambda(A)$ , write  $\mathcal{C}(u) := \{a \in A \mid u \leq_{\mathcal{J}} a\}$  for the *content* of  $u$ . The following restates Theorem 18 in this notation.

► **Corollary 19.** *Let  $f: \Lambda(B) \rightarrow \Lambda(A)$  be a substitution. For any  $v \in \Lambda(B)$ , we have:*

$$\begin{aligned} \uparrow_{\mathcal{J}} f(v) &= \{\varepsilon\} \cup \bigcup_{b \in \mathcal{C}(v)} (\uparrow_{\mathcal{J}} f(b)) \cup \bigcup_{b_1, b_2 \in B} [\uparrow_{\mathcal{L}} f(b_1) \cdot f(b_1^{-1} (\uparrow_{\mathcal{J}} v) b_2^{-1}) \cdot \uparrow_{\mathcal{R}} f(b_2)], \\ \uparrow_{\mathcal{R}} f(v) &= \{\varepsilon\} \cup \bigcup_{b \in B} f((\uparrow_{\mathcal{R}} v) b^{-1}) \cdot (\uparrow_{\mathcal{R}} f(b)), \\ \uparrow_{\mathcal{L}} f(v) &= \{\varepsilon\} \cup \bigcup_{c \in B} (\uparrow_{\mathcal{L}} f(c)) \cdot f(c^{-1} (\uparrow_{\mathcal{L}} v)). \end{aligned}$$

► **Remark.** Several known structural results about  $\widehat{F}_{\mathbf{A}}(A)$  are easy to prove using the saturated-models approach, as one can apply identical arguments to usual combinatorics on words, also see [19, Section 6] or the appendix to this paper. In particular, we can recover the result [20, 3] that  $\widehat{F}_{\mathbf{A}}(A)$  is equidivisible; a monoid  $M$  is called *equidivisible* if for all  $u, v, u', v' \in M$ , if  $uv = u'v'$  then there exists  $x \in M$  such that either  $ux = u'$  and  $xv = v'$ , or  $u'x = u$  and  $xv = v'$ . The pro-aperiodic monoid  $\Lambda(A)$  is equidivisible as well. We also re-prove in this way that the  $\mathcal{L}$ - and  $\mathcal{R}$ -orders on  $\Lambda(A)$  and  $\widehat{F}_{\mathbf{A}}(A)$  are unambiguous, and are total on stabilizers. This was used in [20] to establish the decidability of membership in a number of semidirect products of the form  $\mathbf{V} * \mathbf{A}$ .

## 5 The word problem for omega-terms

In this section, we use our techniques to give an improved proof of the decidability of the word problem for  $\omega$ -terms in  $\widehat{F}_{\mathbf{A}}(A)$ . We begin with a few definitions. An  $\omega$ -term over  $A$  is a term built up from finite words by using concatenation and  $\omega$ -power. If  $M$  is a profinite monoid containing the alphabet  $A$ , then any  $\omega$ -term  $t$  has a natural interpretation  $\llbracket t \rrbracket_M$  in  $M$ , which can be defined inductively using the multiplication and the unary operation  $()^\omega$  on the profinite monoid  $M$ . In the case  $M = \widehat{F}_{\mathbf{A}}(A)$ , we will now inductively define, for any  $\omega$ -term  $t$ , a particular  $A$ -word  $U_t$  in the class  $\llbracket t \rrbracket_{\widehat{F}_{\mathbf{A}}(A)}$ . Let  $\rho$  denote the linear order  $\mathbb{N} + \mathbb{Q} \times \mathbb{Z} + \mathbb{N}^{\text{op}}$ , which is countably saturated (cf. Example 14).

- If  $t$  is a term representing a finite word, let  $U_t$  be that finite word.
- If  $t = t_1 \cdot t_2$ , let  $U_t$  be the  $A$ -word  $U_{t_1} \cdot U_{t_2}$ .
- If  $t = s^\omega$ , let  $U_t$  be the  $A$ -word  $(U_s)^\rho$ .

► **Proposition 20.** *For any  $\omega$ -term  $t$ , the  $A$ -word  $U_t$  is a countably saturated  $A$ -word in the elementary equivalence class  $\llbracket t \rrbracket_{\widehat{F}_{\mathbf{A}}(A)}$ .*

**Proof.** Finite words are countably saturated. Concatenations and  $\rho$ -powers of countably saturated  $A$ -words are clearly countable, and, by Corollary 16,  $\omega$ -saturated. An easy induction, using Theorem 8 for the step involving  $\omega$ -power, shows that  $U_t$  lies in  $\llbracket t \rrbracket_{\widehat{F}_{\mathbf{A}}(A)}$ . ◀

► **Theorem 21.** *For any  $\omega$ -terms  $t_1, t_2$ , the following are equivalent:*

1.  $\llbracket t_1 \rrbracket_{\widehat{F}_{\mathbf{A}}(A)} = \llbracket t_2 \rrbracket_{\widehat{F}_{\mathbf{A}}(A)}$ ,
2.  $U_{t_1}$  is isomorphic to  $U_{t_2}$ .

**Proof.** (2)  $\Rightarrow$  (1) is clear, since isomorphic  $A$ -words are elementarily equivalent. (1)  $\Rightarrow$  (2). By Proposition 20, both  $U_{t_1}$  and  $U_{t_2}$  are countably saturated models in the same elementary equivalence class. By the uniqueness of countably saturated models (cf., e.g., [12, Thm. 2.3.9]),  $U_{t_1}$  and  $U_{t_2}$  are isomorphic. ◀

In order to decide the word problem for  $\omega$ -terms in  $\widehat{F}_{\mathbf{A}}(A)$ , one can now proceed as in [22] and use a decidability procedure for isomorphism of regular words (cf. [8] or [24]) to decide isomorphism of the countably saturated  $A$ -words interpreting the  $\omega$ -terms.

► **Remark.** The original proof of decidability was due to McCammond [26], who introduced normal forms based on an inductively defined notion of rank. The separation of normal forms makes use of his solution to the word problem for free Burnside semigroups of sufficiently large exponent [25], which inspired the definition of the normal forms in the first place. Recently, Almeida, Costa and Zeitoun [6] have provided a new proof that distinct McCammond normal forms represent distinct  $\omega$ -terms in  $\widehat{F}_{\mathbf{A}}(A)$  by introducing star-free languages associated to

normal forms whose closures can be used to separate them. Huschenbett and Kufleitner developed in [22] a new algorithm to solve the word problem for  $\omega$ -terms in  $\widehat{F}_{\mathbf{A}}(A)$  using model-theoretic ideas. They assign the same  $A$ -word that we did above to each  $\omega$ -term. They then prove that two such interpretations of  $\omega$ -terms are elementarily equivalent to each other if and only if they are isomorphic by making use of the non-trivial direction McCammond's normal form theorem. They use work of Bloom and Esik [8] to prove that isomorphism can be decided in exponential time in the size of the expression as an  $\omega$ -term; in the conference presentation of this result they announced that this can be improved to polynomial time using recent work of Lohrey and Mathissen [24]. Here, countably saturated models allow us to give a direct proof of the correctness of the Huschenbett and Kufleitner algorithm, circumventing McCammond's results entirely.

## 6 Factors of omega-terms and well-quasi-orders

In this section and the next, we exploit our model-theoretic view of  $\Lambda(A)$  and  $\widehat{F}_{\mathbf{A}}(A)$  to analyze the  $\mathcal{J}$ -orderings on sets of *factors* of elements in  $\Lambda(A)$ . As special cases of our more general results, we recover several of the structural results on factors of  $\omega$ -terms that were obtained in [4, 5] using McCammond's normal forms. Again, our proofs avoid such normal forms altogether; instead, we use Theorem 18 on factors of a substitution.

We first show how our results in Section 4 give a simple proof of the following result that was proved in an entirely different way by Almeida, Costa, and Zeitoun [5, Theorem 7.4].

► **Theorem 22.** *Prefixes, suffixes and factors of elements in  $\widehat{F}_{\mathbf{A}}(A)$  that are interpretations of  $\omega$ -terms are again interpretations of  $\omega$ -terms.*

**Proof.** We prove the statement for prefixes. We write  $\llbracket t \rrbracket$  as shorthand for  $\llbracket t \rrbracket_{\widehat{F}_{\mathbf{A}}(A)}$ . The statement for suffixes then follows by symmetry, and the statement for factors, in turn, then follows because any factor is a suffix of a prefix. We will prove by induction on the complexity of an  $\omega$ -term  $t$  that, for any  $w \in \widehat{F}_{\mathbf{A}}(A) \setminus \{\varepsilon\}$  such that  $\llbracket t \rrbracket \leq_{\mathcal{R}} w$ , we have  $w = \llbracket s \rrbracket$  for some  $\omega$ -term  $s$ . Prefixes of a finite word are finite words. If  $t = t_0 \cdot t_1$  for some  $\omega$ -terms  $t_0$  and  $t_1$ , and  $\llbracket t \rrbracket \leq_{\mathcal{R}} w$ , we apply Theorem 18.2 in the special case of a concatenation, which we recall is the substitution of  $u_0 := \llbracket t_0 \rrbracket$  and  $u_1 := \llbracket t_1 \rrbracket$  into the  $\{0, 1\}$ -word  $v := 01$ . If  $\llbracket t_0 \rrbracket \leq_{\mathcal{R}} w$ , then we are done immediately by induction. Otherwise, we have  $w = \llbracket t_0 \rrbracket \cdot v$  for some  $v$  with  $\llbracket t_1 \rrbracket \leq_{\mathcal{R}} v$ . By induction, pick an  $\omega$ -term  $s'$  such that  $\llbracket s' \rrbracket = v$ . Then for the  $\omega$ -term  $s := t_0 s'$ , we have  $\llbracket s \rrbracket = w$ . If  $t = r^\omega$  for some  $\omega$ -term  $r$ , and  $\llbracket t \rrbracket \leq_{\mathcal{R}} w$ , we apply Theorem 18.2 in the special case of an  $\omega$ -power, which we recall is the substitution of  $u := \llbracket r \rrbracket$  into the  $\{b\}$ -word  $v := b^\omega$ . There exist  $z \in \widehat{F}_{\mathbf{A}}(1) = \mathbb{N} \cup \{\omega\}$  and  $x \in \widehat{F}_{\mathbf{A}}(A)$  with  $\llbracket r \rrbracket \leq_{\mathcal{R}} x$  such that  $w = r^z x$ . By the induction hypothesis, pick an  $\omega$ -term  $s'$  such that  $\llbracket s' \rrbracket = x$ . The  $\omega$ -term  $s := r^z s'$  gives  $\llbracket s \rrbracket = w$ . ◀

► **Remark.** Almeida, Costa and Zeitoun also proved that interpretations of  $\omega$ -terms have only finitely many regular  $\mathcal{J}$ -classes above them. In [19, Section 8], we easily prove this result using our new method; we omit this here for reasons of space.

Recall that a *well-quasi-order* (wqo) is a quasi-order that does not contain infinite antichains or infinite descending chains; equivalently, for every infinite sequence  $(q_i)_{i \in \omega}$  in  $Q$ , there exist  $i < j$  such that  $q_i \preceq q_j$  (see e.g. [31, §10.3]). We will call an element  $u \in \Lambda(A)$  *well-factor-ordered* (wfo) if the reverse  $\mathcal{J}$ -order,  $\geq_{\mathcal{J}}$ , is a well-quasi-order on the set  $\uparrow_{\mathcal{J}} u$  of factors of  $u$ . Similarly, we call  $u \in \Lambda(A)$  *well-prefix-ordered* (wpo) if  $\geq_{\mathcal{R}}$  is a wqo on  $\uparrow_{\mathcal{R}} u$  and *well-suffix-ordered* (wso) if  $\geq_{\mathcal{L}}$  is a wqo on  $\uparrow_{\mathcal{L}} u$ .

► **Remark.** Note that, by Proposition 9,  $\widehat{F}_{\mathbf{A}}(A)$  is upward closed in  $\Lambda(A)$  with respect to the  $\mathcal{J}$ -,  $\mathcal{R}$ - and  $\mathcal{L}$ -orders, so that the following results apply immediately to  $\widehat{F}_{\mathbf{A}}(A)$ , as well.

The proof of the following theorem uses Theorem 18 to analyze the factors of the result of substituting wfo elements into wfo elements.

► **Theorem 23.** *Let  $f: \Lambda(B) \rightarrow \Lambda(A)$  be a substitution. If  $f(b)$  is well-factor-ordered for each  $b \in B$ , and  $v \in \Lambda(B)$  is well-factor-ordered, then  $f(v)$  is well-factor-ordered.*

Analogous results, with simpler proofs, hold for well-prefix-ordered and well-suffix-ordered elements.

► **Theorem 24.** *Let  $f: \Lambda(B) \rightarrow \Lambda(A)$  be a substitution. If  $f(b)$  is well-prefix-ordered for each  $b \in B$ , and  $v \in \Lambda(B)$  is well-prefix-ordered, then  $f(v)$  is well-prefix-ordered.*

By symmetry, we have the following corollary.

► **Corollary 25.** *Let  $f: \Lambda(B) \rightarrow \Lambda(A)$  be a substitution. If  $f(b)$  is well-suffix-ordered for each  $b \in B$ , and  $v \in \Lambda(B)$  is well-suffix-ordered, then  $f(v)$  is well-suffix-ordered.*

The following special case of the above three results recovers [5, Corollary 5.6] and [5, Theorem 7.3].

► **Corollary 26.** *The sets of well-factor-ordered, well-prefix-ordered, and well-suffix-ordered elements in  $\Lambda(A)$  are closed under concatenation and  $\omega$ -power. In particular, interpretations of  $\omega$ -terms in  $\widehat{F}_{\mathbf{A}}(A)$  are well-factor-ordered.*

We end this section by showing that there are many more well-factor-ordered elements in  $\widehat{F}_{\mathbf{A}}(A)$  and  $\Lambda(A)$  than just the interpretations of  $\omega$ -terms. Almeida showed [2, Theorem 2.6] that  $\widehat{F}_{\mathbf{A}}(A) \setminus A^*$  contains maximal elements with respect to the  $\mathcal{J}$ -ordering and that they correspond in a sense that can be made precise to uniformly recurrent words. Moreover, it is shown in [5, Proposition 3.2] that every element of  $\widehat{F}_{\mathbf{A}}(A) \setminus A^*$  is  $\mathcal{J}$ -below a maximal element. Notice that  $\omega$  is the unique maximal element of  $\widehat{F}_{\mathbf{A}}(1) \setminus \mathbb{N}$ . The  $\mathcal{J}$ -maximal elements of  $\widehat{F}_{\mathbf{A}}(A) \setminus A^*$  are in particular  $\mathcal{J}$ -maximal in  $\Lambda(A) \setminus A^*$ , and all of the latter elements are well-factor-ordered:

► **Proposition 27.** *Let  $w$  be a maximal element of  $\Lambda(A) \setminus A^*$  in the  $\mathcal{J}$ -order. Then  $w$  is well-factor-ordered.*

It follows that the smallest submonoid of  $\widehat{F}_{\mathbf{A}}(A)$  containing all finite words and  $\mathcal{J}$ -maximal elements that is closed under the  $\omega$ -power consists of well-factor-ordered elements. Also, substituting  $\omega$ -terms into  $\mathcal{J}$ -maximal elements will provide new example of well-factor-ordered elements.

## 7 Regular languages of factors

We call an element  $u \in \Lambda(A)$  *factor-regular* if the set  $F(u) := \uparrow_{\mathcal{J}}u \cap A^*$  of finite factors of  $u$  is a regular language. We call  $u$  *prefix-regular* if  $P(u) := \uparrow_{\mathcal{R}}u \cap A^*$  is a regular language, and *suffix-regular* if  $S(u) := \uparrow_{\mathcal{L}}u \cap A^*$  is a regular language. If  $u \in \Lambda(A) \setminus A^*$  is prefix-regular, then so is any element of  $u\Lambda(A)$ , and dually for suffix-regular elements. We call a substitution  $f: \Lambda(B) \rightarrow \Lambda(A)$  *non-erasing* if  $f(b) \neq \varepsilon$  for every  $b \in B$ . Notice that a non-erasing substitution  $f$  sends the ideal  $\Lambda(B) \setminus B^*$  to the ideal  $\Lambda(A) \setminus A^*$ .

In the following theorem, which is another application of Theorem 18, we prove that prefix-regularity and suffix-regularity are stable under non-erasing substitutions, and so is factor-regularity under certain additional assumptions.

► **Theorem 28.** *Let  $f: \Lambda(B) \rightarrow \Lambda(A)$  be a non-erasing substitution and  $v \in \Lambda(B)$ .*

1. *If  $f(b)$  is prefix-regular, for each  $b \in B$ , and  $v$  is prefix-regular, then  $f(v)$  is prefix-regular.*
2. *If  $f(b)$  is suffix-regular, for each  $b \in B$ , and  $v$  is suffix-regular, then  $f(v)$  is suffix-regular.*
3. *If  $f(b)$  is factor-regular, prefix-regular and suffix-regular, for each  $b \in B$ , and  $v$  is factor-regular, then  $f(v)$  is factor-regular.*

**Proof.** We only prove (3), referring to the appendix for the proofs of (1) and (2). Put  $C = \{b \in B^* \mid f(b) \in A^*\}$ . Using Corollary 19, it is straightforward to verify that

$$F(f(v)) = \bigcup_{b \in B \cap F(v)} F(f(b)) \cup \bigcup_{b_1, b_2 \in B \cup \{\varepsilon\}} S(f(b_1))f(C^* \cap b_1^{-1}F(w)b_2^{-1})P(f(b_1))$$

and so the desired result follows from closure of regular languages under boolean operations, product, left and right quotients and homomorphic image. ◀

For a finite alphabet  $A$  and  $b \notin A$ , let  $f: \Lambda(A \cup \{b\}) \rightarrow \Lambda(A)$  be the substitution erasing  $b$  and fixing  $A$ . Note that if  $v \in \Lambda(A)$  is not prefix-regular, then  $b^\omega v$  is prefix-regular, but  $f(b^\omega v) = v$  is not. Thus, the assumption in Theorem 28 that  $f$  is non-erasing is necessary.

The next corollary recovers [5, Corollary 7.6].

► **Corollary 29.** *Interpretations of  $\omega$ -terms are prefix-, suffix- and factor-regular.*

► **Remark.** There are more factor-regular elements than just interpretations of  $\omega$ -terms. In particular, the minimal ideal  $I$  of  $\widehat{F}_{\mathbf{A}}(A)$  consists of those elements containing every finite word as a factor, so every element of  $I$  is factor-regular. Thus if we substitute  $\omega$ -terms over  $B$  into an element of the minimal ideal of  $\widehat{F}_{\mathbf{A}}(A)$ , then we obtain factor-regular elements of  $\widehat{F}_{\mathbf{A}}(B)$ . It is not the case that every element of the minimal ideal of  $\widehat{F}_{\mathbf{A}}(A)$  is prefix-regular or suffix-regular. In fact, it is easy to see using the pumping lemma that  $w \in \Lambda(A) \setminus A^*$  is prefix-regular if and only if  $w = uv^\omega z$  with  $u, v$  finite and  $v \neq \varepsilon$ . A dual description holds for suffix-regular elements.

## 8 Conclusion

In this paper we gave a new approach to the free pro-aperiodic monoid by viewing its elements as elementary equivalence classes of pseudofinite words. This view led us to consider  $\widehat{F}_{\mathbf{A}}(A)$  as a closed submonoid of the larger monoid  $\Lambda(A)$  consisting of elementary equivalence classes of arbitrary  $A$ -words. The model-theoretic fact that each such class contains an  $\omega$ -saturated model enabled us to analyze factors in  $\Lambda(A)$  combinatorially. Thus, we substantiate the claim made in [20] that one may “transfer arguments from Combinatorics on Words to the profinite context”: our approach using saturated models makes this idea precise.

The newly identified pro-aperiodic monoid  $\Lambda(A)$  poses several interesting questions for future work. In particular, it would be interesting to study the algebraic structure of  $\Lambda(A)$  in more detail. Here, connections with the work of Carton, Colcombet and Puppis on algebras for words over countable linear orderings [11] are to be expected.

We also plan to explore in future work how this method might be extended to other profinite monoids. In particular, one could try to analyze the absolutely free profinite monoid in this way, but this would require replacing first-order model theory by monadic second-order model theory. In this direction, we foresee connections to Shelah’s seminal work [33].

In a different direction, we hope that our approach could be useful for more easily analyzing aperiodic pointlike sets and related notions, in particular because a logical approach has recently proved useful for deciding problems in the first-order quantifier alternation hierarchy [29].



## References

- 1 J. Almeida. *Finite semigroups and universal algebra*, volume 3 of *Series in Algebra*. World Scientific Publishing Co. Inc., River Edge, NJ, 1994. Translated from the 1992 Portuguese original and revised by the author.
- 2 J. Almeida. Profinite groups associated with weakly primitive substitutions. *Fundam. Prikl. Mat.*, 11(3):13–48, 2005. doi:10.1007/s10958-007-0242-y.
- 3 J. Almeida and A. Costa. Infinite-vertex free profinite semigroupoids and symbolic dynamics. *J. Pure Appl. Algebra*, 213(5):605–631, 2009. doi:10.1016/j.jpaa.2008.08.009.
- 4 J. Almeida, J. C. Costa, and M. Zeitoun. Some structural properties of the free profinite aperiodic semigroup. In *2nd AutoMathA Conference*, 2009.
- 5 J. Almeida, J. C. Costa, and M. Zeitoun. Iterated periodicity over finite aperiodic semigroups. *European J. Combin.*, 37:115–149, 2014. doi:10.1016/j.ejc.2013.07.011.
- 6 J. Almeida, J. C. Costa, and M. Zeitoun. McCammond’s normal forms for free aperiodic semigroups revisited. *LMS J. Comput. Math.*, 18(1):130–147, 2015. doi:10.1112/S1461157014000448.
- 7 J. Almeida and M. V. Volkov. Subword complexity of profinite words and subgroups of free profinite semigroups. *Internat. J. Algebra Comput.*, 16(2):221–258, 2006. doi:10.1142/S0218196706002883.
- 8 S. L. Bloom and Z. Ésik. The equational theory of regular words. *Information and Computation*, 197:55–89, 2005.
- 9 M. Bojańczyk. Recognisable Languages over Monads. arXiv:1502.04898v1, 2015.
- 10 J. A. Brzozowski. Hierarchies of aperiodic languages. *Theoret. Informatics Appl.*, 10(2):33–49, 1976.
- 11 O. Carton, T. Colcombet, and G. Puppis. Regular languages of words over countable linear orderings. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 6756 of *LNCS*, pages 125–136. Springer, 2011.
- 12 C.-C. Chang and J. H. Keisler. *Model Theory*. North-Holland, Amsterdam-London, third edition, 1990.
- 13 H. C. Doets. *Completeness And Definability : Applications Of The Ehrenfeucht Game In Second-Order And Intensional Logic*. PhD thesis, Universiteit van Amsterdam, May 1987. URL: <http://oai.cwi.nl/oai/asset/22742/22742A.pdf>.
- 14 S. Eilenberg. *Automata, languages, and machines. Vol. B*. Academic Press, New York, 1976. With two chapters (“Depth decomposition theorem” and “Complexity of semigroups and morphisms”) by Bret Tilson, Pure and Applied Mathematics, Vol. 59.
- 15 M. Gehrke. Stone duality, topological algebra, and recognition. *J. Pure Appl. Algebra*, 220(7):2711–2747, 2016.
- 16 M. Gehrke, S. Grigorieff, and J.-É. Pin. Duality and equational theory of regular languages. In L. Aceto et al., editor, *ICALP 2008, Part II*, number 5126 in *Lect. Notes Comput. Sci.*, pages 246–257. Springer, 2008.
- 17 M. Gehrke, A. Krebs, and J.-É. Pin. Ultrafilters on words for a fragment of logic. *Theoretical Computer Science*, 610 A:37–58, 2016.
- 18 M. Gehrke, D. Petrisan, and L. Reggio. The Schützenberger product for syntactic spaces. In *43rd International Colloquium on Automata, Languages and Programming (ICALP 2016)*, 2016.
- 19 S. J. v. Gool and B. Steinberg. Pro-aperiodic monoids and model theory. Technical report, 2016. URL: <http://www.samvangool.net/papers/goolsteinberg2016TR.pdf>.
- 20 K. Henckell, J. Rhodes, and B. Steinberg. A profinite approach to stable pairs. *Internat. J. Algebra Comput.*, 20(2):269–285, 2010. doi:10.1142/S0218196710005650.
- 21 W. Hodges. *Model theory*, volume 42 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 1993.

- 22 M. Huschenbett and M. Kuffeitner. Ehrenfeucht-Fraïssé Games on Omega-Terms. In *STACS*, pages 374–385, 2014.
- 23 K. Krohn and J. Rhodes. Complexity of finite semigroups. *Ann. of Math. (2)*, 88:128–160, 1968.
- 24 M. Lohrey and C. Mathissen. Isomorphism of regular trees and words. *Inform. and Comput.*, 224:71–105, 2013. doi:10.1016/j.ic.2013.01.002.
- 25 J. P. McCammond. The solution to the word problem for the relatively free semigroups satisfying  $T^a = T^{a+b}$  with  $a \geq 6$ . *Internat. J. Algebra Comput.*, 1(1):1–32, 1991. doi:10.1142/S021819679100002X.
- 26 J. P. McCammond. Normal forms for free aperiodic semigroups. *Int. J. Algebra Comput.*, 11(5):581–625, 2001.
- 27 R. McNaughton and S. Papert. *Counter-free automata*. Number 65 in M.I.T. Research Monograph. M.I.T. Press, 1971.
- 28 J.-É. Pin. The dot-depth hierarchy, 45 years later. Preprint available from the author’s homepage.
- 29 T. Place and M. Zeitoun. Going Higher in the First-Order Quantifier Alternation Hierarchy on Words. In *Proceedings ICALP*, 2014.
- 30 J. Rhodes and B. Steinberg. *The q-theory of Finite Semigroups*. Springer, 2009.
- 31 J. G. Rosenstein. *Linear orderings*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers], 1982.
- 32 M.-P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.
- 33 S. Shelah. The monadic theory of order. *Ann. of Math. (2)*, 102(3):379–419, 1975.
- 34 M. H. Stone. The Theory of Representation for Boolean Algebras. *Trans. Amer. Math. Soc.*, 74(1):37–111, 1936.
- 35 H. Straubing. *Finite automata, formal logic, and circuit complexity*. Progress in Theoretical Computer Science. Birkhäuser Boston Inc., Boston, 1994.
- 36 H. Straubing and P. Weil. Varieties. In J.-É. Pin, editor, *Handbook of automata theory*. European Mathematical Society Publishing House, to appear. ArXiv:1502.03951.
- 37 J. Väänänen. Pseudo-finite model theory. *Mat. Contemp.*, 24:169–183, 2003.

# Trimming and Gluing Gray Codes<sup>\*†</sup>

Petr Gregor<sup>1</sup> and Torsten Mütze<sup>2</sup>

- 1 Department of Theoretical Computer Science and Mathematical Logic,  
Charles University, Praha, Czech Republic  
gregor@ktiml.mff.cuni.cz
- 2 Institut für Mathematik, TU Berlin, Berlin, Germany  
muetze@math.tu-berlin.de

---

## Abstract

We consider the algorithmic problem of generating each subset of  $[n] := \{1, 2, \dots, n\}$  whose size is in some interval  $[k, l]$ ,  $0 \leq k \leq l \leq n$ , exactly once (cyclically) by repeatedly adding or removing a single element, or by exchanging a single element. For  $k = 0$  and  $l = n$  this is the classical problem of generating all  $2^n$  subsets of  $[n]$  by element additions/removals, and for  $k = l$  this is the classical problem of generating all  $\binom{n}{k}$  subsets of  $[n]$  by element exchanges. We prove the existence of such cyclic minimum-change enumerations for a large range of values  $n$ ,  $k$ , and  $l$ , improving upon and generalizing several previous results. For all these existential results we provide optimal algorithms to compute the corresponding Gray codes in constant time  $\mathcal{O}(1)$  per generated set and space  $\mathcal{O}(n)$ . Rephrased in terms of graph theory, our results establish the existence of (almost) Hamilton cycles in the subgraph of the  $n$ -dimensional cube  $Q_n$  induced by all levels  $[k, l]$ . We reduce all remaining open cases to a generalized version of the middle levels conjecture, which asserts that the subgraph of  $Q_{2k+1}$  induced by all levels  $[k - c, k + 1 + c]$ ,  $c \in \{0, 1, \dots, k\}$ , has a Hamilton cycle. We also prove an approximate version of this conjecture, showing that this graph has a cycle that visits a  $(1 - o(1))$ -fraction of all vertices.

**1998 ACM Subject Classification** G.2.1 Combinatorics, G.2.2 Graph Theory

**Keywords and phrases** Gray code, subset, combination, loopless algorithm

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.40

## 1 Introduction

Generating all objects in a combinatorial class such as permutations, subsets, combinations, partitions, trees, strings etc. is one of the oldest and most fundamental algorithmic problems, and such generation algorithms appear as core building blocks in a wide range of practical applications (see the survey [30]). In fact, half of the most recent volume [21] of Donald Knuth's seminal series *The Art of Computer Programming* is devoted entirely to this fundamental subject. The ultimate goal for algorithms that efficiently generate each object of a particular combinatorial class exactly once is to generate each new object in constant time, which is best possible. Such optimal algorithms are sometimes called *loopless algorithms*, a term coined by Ehrlich in his influential paper [10]. Note that a constant-time algorithm requires in particular that consecutively generated objects differ only in a constant amount, e.g., in a single transposition of a permutation, in adding or removing a single element from a set, or in a single tree rotation operation. These types of orderings have become known as

---

\* A full version of the paper is available at <https://arxiv.org/abs/1607.08806>.

† The first author was supported by the Czech Science Foundation grant GA14-10799S.



© Petr Gregor and Torsten Mütze;

licensed under Creative Commons License CC-BY

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 40; pp. 40:1–40:14

Leibniz International Proceedings in Informatics



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

*combinatorial Gray codes.* Here are two fundamental examples for this kind of generation problems: (1) The so-called *reflected Gray code* is a method to generate all  $2^n$  many subsets of  $[n] := \{1, 2, \dots, n\}$  by repeatedly adding or removing a single element. It is named after Frank Gray, a physicist and researcher at Bell Labs, and appears in his patent [13]. The reflected Gray code has many interesting properties (see [21, Section 7.2.1.1]), and there is a simple loopless algorithm to compute it [2, 10]. (2) Of similar importance in practice is the problem of generating all  $\binom{n}{k}$  many  $k$ -element subsets of  $[n]$  by repeatedly exchanging a single element. Also for this problem, loopless algorithms are well-known [2, 4, 8, 9, 10, 18, 28, 35] (see also [21, Section 7.2.1.3]).

In this work we consider far-ranging generalizations of the classical problems (1) and (2). Specifically, we consider the algorithmic problem of generating all (or almost all) subsets of  $[n]$  whose size is in some interval  $[k, l]$ ,  $0 \leq k \leq l \leq n$ , by repeatedly adding or removing a single element, or by exchanging a single element (this will be made more precisely in a moment). We recover the classical problems (1) and (2) mentioned before as special cases by setting  $k = 0$  and  $l = n$ , or by setting  $k = l$ , respectively. It turns out that the entire parameter range in between these special cases offers plenty of room for surprising discoveries and hard research problems (take a peek at the known results in Figure 1 below).

In a computer a subset of  $[n]$  is conveniently represented by the corresponding characteristic bitstring  $x$  of length  $n$ , where all the 1-bits of  $x$  correspond to the elements contained in the set, and the 0-bits to the elements not contained in the set. The before-mentioned subset generation problems can thus be rephrased as Hamilton cycle problems in subgraphs of the *cube*  $Q_n$ , the graph that has as vertices all bitstrings of length  $n$ , with an edge between any two vertices (=bitstrings) that differ in exactly one bit. We refer to the number of 1-bits in a bitstring  $x$  as the *weight of  $x$* , and we refer to the vertices of  $Q_n$  with weight  $k$  as the  *$k$ -th level of  $Q_n$*  (there are  $\binom{n}{k}$  vertices on level  $k$ ). Moreover, we let  $Q_{n,[k,l]}$ ,  $0 \leq k \leq l \leq n$ , denote the subgraph of  $Q_n$  induced by all levels  $[k, l]$ . In terms of sets, the vertices of the cube  $Q_n$  correspond to subsets of  $[n]$ , and flipping a bit along an edge corresponds to adding or removing a single element. The weight of a bitstring corresponds to the size of the set, and the vertices on level  $k$  correspond to all  $k$ -element subsets of  $[n]$ .

One of the hard instances of the before-mentioned general enumeration problem in  $Q_{n,[k,l]}$  is when  $n = 2k + 1$  and  $l = k + 1$ . The existence of a Hamilton cycle in the graph  $Q_{2k+1,[k,k+1]}$  for any  $k \geq 1$  is asserted by the well-known *middle levels conjecture*, raised independently in the 80's by Havel [16] and Buck and Wiedemann [3]. The conjecture has also been attributed to Dejter, Erdős, Trotter [20] and various others, and also appears in the popular books [5, 21, 36]. The middle levels conjecture has attracted considerable attention over the last 30 years [6, 7, 12, 15, 17, 19, 20, 27, 29, 31, 32, 33], and a positive solution, an existence proof for a Hamilton cycle in  $Q_{2k+1,[k,k+1]}$  for any  $k \geq 1$ , has been announced only recently.

► **Theorem 1** ([23]). *For any  $k \geq 1$ , the graph  $Q_{2k+1,[k,k+1]}$  has a Hamilton cycle.*

The following generalization of the middle levels conjecture was proposed in [15].

► **Conjecture 2** ([15]). *For any  $k \geq 1$  and  $c \in \{0, 1, \dots, k\}$ , the graph  $Q_{2k+1,[k-c,k+1+c]}$  has a Hamilton cycle.*

Conjecture 2 clearly holds for all  $k \geq 1$  and  $c = k$  as  $Q_{2k+1,[0,2k+1]} = Q_{2k+1}$  (this is problem (1) from before). It is known that the conjecture also holds for all  $k \geq 1$  and  $c = k - 1$  [11, 22] and  $c = k - 2$  [15]. By Theorem 1 it also holds for all  $k \geq 1$  and  $c = 0$ .

Another generalization of Theorem 1 in a slightly different direction (still a special case in our general framework) is the following result.

► **Theorem 3** ([26]). *For any  $n \geq 3$  and  $k \in \{1, 2, \dots, n - 2\}$ , the graph  $Q_{n,[k,k+1]}$  has a cycle that visits all vertices in the smaller bipartite class.*

The idea for the proof of Theorem 3 (induction over  $n$ ) was first presented in [16]. In that paper, the theorem was essentially proved conditional on the validity of the hardest case  $n = 2k + 1$ , the middle levels conjecture, which was established as a theorem (Theorem 1) only much later. In [26], Theorem 3 was proved unconditionally, and the proof technique was refined further to also prove Hamiltonicity results for the so-called bipartite Kneser graphs, another generalization of the middle levels conjecture.

Conjecture 2 and Theorem 3 immediately suggest the following common generalization: For which intervals  $[k, l]$  does the cube  $Q_{n,[k,l]}$  have a Hamilton cycle? The graph  $Q_{n,[k,l]}$  is bipartite (the two partition classes are given by the parity of the number of 1-bits of the vertices), and it is clear that a Hamilton cycle can exist only if the two partition classes have the same size, which happens only for odd dimension  $n$  and between two symmetric levels  $k$  and  $l = n - k$  (Conjecture 2). However, we may slightly relax this question, and ask for a long cycle. To this end, we denote for any bipartite graph  $G$  by  $v(G)$  the number of vertices of  $G$ , and by  $\delta(G)$  the difference between the larger and the smaller partition class. Note that in any bipartite graph  $G$  (as  $Q_{n,[k,l]}$ ) the length of any cycle is at most  $v(G) - \delta(G)$  (i.e., the cycle visits all vertices in the smaller partition class). We call such a cycle a *saturating cycle*. Observe that if both partition classes have the same size (i.e.  $\delta(G) = 0$ ), then a saturating cycle is a Hamilton cycle. Hence saturating cycles naturally generalize Hamilton cycles for unbalanced bipartite graphs. The right common generalization of Conjecture 2 and Theorem 3 therefore is:

► **Question 4.** For which intervals  $[k, l]$  does the cube  $Q_{n,[k,l]}$  have a saturating cycle?

A saturating cycle necessarily omits some vertices (exactly  $\delta(Q_{n,[k,l]})$  many) from the larger bipartite class. However, if we insist on all vertices of  $Q_{n,[k,l]}$  to be included in a cycle, then this can be achieved by allowing steps where instead of only a single bitflip, two bits are flipped (the underlying graph  $Q_{n,[k,l]}$  is augmented by adding distance-2 edges). In this case we may ask for a (cyclic) enumeration of all vertices of  $Q_{n,[k,l]}$  that minimizes the number of these ‘cheating’ distance-2 steps, i.e., for an enumeration that has only  $\delta(Q_{n,[k,l]})$  many distance-2 steps (this is clearly the least number one can hope for). We call such an enumeration a *tight enumeration*. A tight enumeration can be seen as a travelling salesman tour of length  $v(Q_{n,[k,l]}) + \delta(Q_{n,[k,l]})$  through all vertices of  $Q_{n,[k,l]}$ , where distances are measured by Hamming distance (=number of bitflips). We may ask in full generality:

► **Question 5.** For which intervals  $[k, l]$  is there a tight enumeration of the vertices of  $Q_{n,[k,l]}$ ?

If both partition classes of  $Q_{n,[k,l]}$  have the same size (i.e.  $\delta(Q_{n,[k,l]}) = 0$ ), then a tight enumeration is a Hamilton cycle in this graph. Note also that Question 5 is a sweeping generalization of the following well-known result regarding problem (2) mentioned before.

► **Theorem 6** ([35]). *For any  $n \geq 2$  and  $1 \leq k \leq n - 1$  there is a cyclic enumeration of all weight  $k$  bitstrings of length  $n$  such that any two consecutive bitstrings differ in exactly 2 bits.*

In fact, several of the subsequently mentioned results of this paper will be proved by extending the original approach from [35] to prove Theorem 6.

## 1.1 Our results

In this work we answer Question 4 and Question 5 for a large range of values  $n$ ,  $k$  and  $l$ . The different ranges of parameters covered by our results are illustrated in Figure 1.

Moreover, we provide optimal algorithms to compute the corresponding saturating cycles/tight enumerations.

Our first set of results resolves Question 4 positively for all possible values of  $k$  and  $l$  except the cases covered by Conjecture 2 (the case  $l = k + 1$  is already covered by Theorem 3), see the left-hand side of Figure 1.

► **Theorem 7.** *For any  $n \geq 3$  the graph  $Q_{n,[k,l]}$  has a saturating cycle in the following cases:*

- (i) *If  $0 = k < l \leq n$  or  $0 \leq k < l = n$  and  $l - k \geq 2$ .*
- (ii) *If  $1 \leq k < l \leq n - 1$  and  $l - k \geq 2$  is even.*
- (iii) *If  $1 \leq k < l \leq \lceil n/2 \rceil$  or  $\lfloor n/2 \rfloor \leq k < l \leq n - 1$  and  $l - k \geq 3$  is odd.*
- (iv) *If  $1 \leq k < l \leq n - 1$  and  $l - k \geq 3$  is odd, under the additional assumption that  $Q_{2m+1,[m-c,m+1+c]}$ ,  $c := (l - k - 1)/2$ , has a Hamilton cycle for all  $m = c, c + 1, \dots, (\min(k + l, 2n - k - l) - 1)/2$ .*

Our second set of results resolves Question 5 positively for all possible values of  $k$  and  $l$  except the cases covered by Conjecture 2 (the case  $l = k$  is already covered by Theorem 6), see the right-hand side of Figure 1.

► **Theorem 8.** *For any  $n \geq 3$  there is a tight enumeration of the vertices of  $Q_{n,[k,l]}$  in the following cases:*

- (i) *If  $0 = k < l \leq n$  or  $0 \leq k < l = n$ .*
- (ii) *If  $1 \leq k < l \leq n$  and  $l - k \geq 2$  is even.*
- (iii) *If  $1 \leq k < l \leq n - 1$  and  $l - k = 1$ .*
- (iv) *If  $1 \leq k < l \leq \lceil n/2 \rceil$  or  $\lfloor n/2 \rfloor \leq k < l \leq n - 1$  and  $l - k \geq 3$  is odd.*
- (v) *If  $1 \leq k < l \leq n - 1$  and  $l - k \geq 3$  is odd, under the additional assumption that  $Q_{2m+1,[m-c,m+1+c]}$ ,  $c := (l - k - 1)/2$ , has a Hamilton cycle for all  $m = c, c + 1, \dots, (\min(k + l, 2n - k - l) - 1)/2$ .*

Note that the last part (iv) of Theorems 7 and 8 is conditional on the validity of Conjecture 2. In fact, by what we said before (recall the paragraph below Conjecture 2) we know that the additional assumption in (iv) is satisfied for  $m \in \{c, c + 1, c + 2\}$  (so the statement could be slightly strengthened).

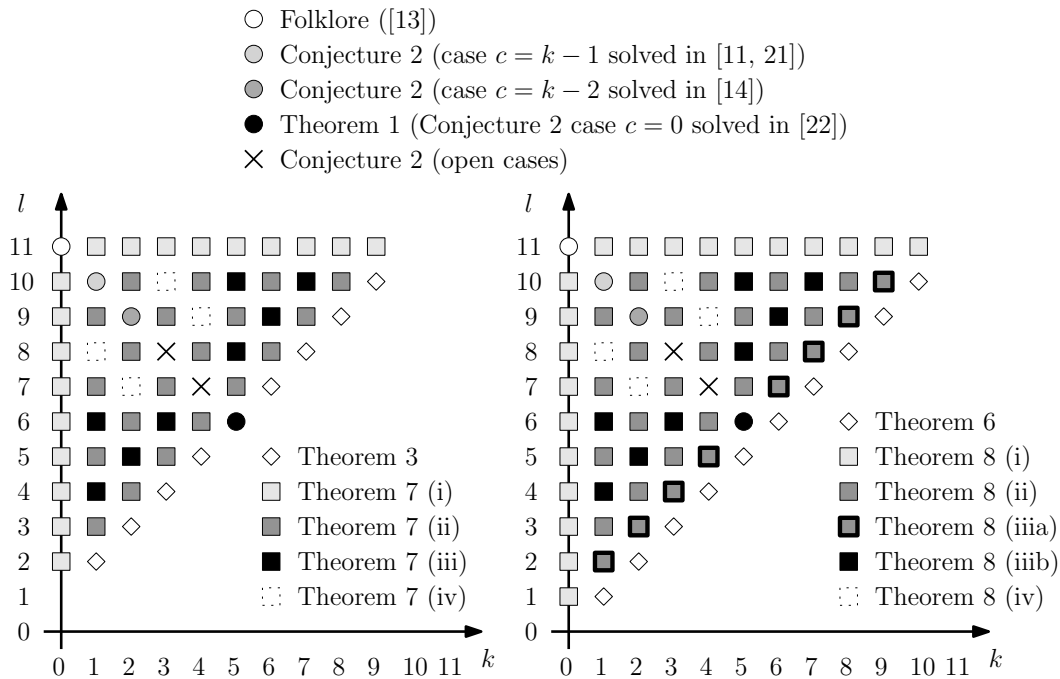
The tight enumerations we construct to prove Theorem 8 have the additional property that all distance-2 steps are within single levels (but never between two different levels  $k$  and  $k + 2$ ). In terms of sets, these steps therefore correspond to exchanging a single element.

For all the unconditional results in Theorems 7 and 8 we provide corresponding optimal generation algorithms.

► **Theorem 9.**

- (a) *For any interval  $[k, l]$  as in case (i) or (ii) of Theorems 7 and 8, respectively, there is a corresponding loopless algorithm that generates each bitstring of a saturating cycle or a tight enumeration of the vertices of  $Q_{n,[k,l]}$  in time  $\mathcal{O}(1)$ .*
- (b) *For any interval  $[k, l]$  as in case (iii) of Theorems 7 and 8, respectively, there is a corresponding algorithm that generates each bitstring of a saturating cycle or a tight enumeration of the vertices of  $Q_{n,[k,l]}$  in time  $\mathcal{O}(1)$  on average.*

It should be noted that the algorithms for part (a) of Theorem 9 are considerably simpler than those for part (b). The reason is that the underlying constructions are entirely different. In particular, for part (b) we repeatedly call the (average) constant-time algorithm to compute a Hamilton cycle in  $Q_{2m+1,[m,m+1]}$ ,  $m \leq \lfloor (n - 1)/2 \rfloor$ , an algorithmic version of Theorem 1,



**Figure 1** The different cases of  $k$  and  $l$  covered by our two main theorems on saturating cycles (left, Theorem 7) and on tight enumerations (right, Theorem 8) in  $Q_{n,[k,l]}$  for the case  $n = 11$ . A more extensive animation of the entire parameter space  $(n, k, l)$  is available on the second author’s website [1].

presented in [24, 25] (and this algorithm is admittedly rather complex). The initialization time of our algorithms is  $\mathcal{O}(n)$ , and the required space is  $\mathcal{O}(n)$ .

We implemented all these algorithms in C++, and we invite the reader to experiment with this code, which can be found on our website [1].

In view of these results, the only remaining (and therefore even more interesting) open case is the question whether the cube of odd dimension has a Hamilton cycle between any two symmetric levels, i.e., Conjecture 2 (these open cases are represented by crosses in Figure 1). Given the results from [15] and [23], the next natural step towards resolving this conjecture would be to investigate whether the graphs  $Q_{2k+1,[3,2k-2]}$  or  $Q_{2k+1,[k-1,k+2]}$  have a Hamilton cycle for all  $k \geq 1$ .

In this work we provide the following partial result towards the general conjecture: We show the existence of long cycles in the graph  $Q_{2k+1,[k-c,k+1+c]}$ ,  $c \in \{0, 1, \dots, k\}$ . This approximate version of the conjecture is similar in spirit to the line of work [12, 19, 29, 31] that preceded the proof of Theorem 1.

► **Theorem 10.** *For any  $k \geq 1$  and  $c \in \{0, 1, \dots, k\}$ , the graph  $Q_{2k+1,[k-c,k+1+c]}$  has a cycle that visits at least a  $(1 - \epsilon)$ -fraction of all vertices, where  $\epsilon := \frac{1}{2(c+1)} \min(1, \exp(\frac{(c+1)^2}{k-c}) - 1)$ . In particular, for any  $c$  and  $k \rightarrow \infty$ , the cycle visits a  $(1 - o(1))$ -fraction of all vertices.*

### 1.2 Related work

In [10] an algorithm is presented that generates the vertices of  $Q_{n,[k,l]}$  (for an arbitrary interval  $[k, l]$ ) such that any two consecutive vertices have Hamming distance 1 or 2, where the value 2 appears only between vertices on level  $k$  and  $l$ , but the Hamming distance between

the first and last vertex is arbitrary (possibly  $n$ ). The running time of this algorithm is  $\mathcal{O}(n)$  per generated vertex. In addition, this paper presents a loopless algorithm (time  $\mathcal{O}(1)$  per vertex) to generate all vertices in  $Q_{n,[k,l]}$  level by level, using only distance-2 steps in each level. In particular, these enumerations are not cycles in  $Q_{n,[k,l]}$ , and they are not tight.

In [34] the authors present algorithms for enumerating all vertices of  $Q_{n,[k,l]}$  (for an arbitrary interval  $[k,l]$ ) such that any two consecutive bitstrings have Levenshtein distance at most 2 and Hamming distance at most 4. The Levenshtein distance is the minimum number of bit insertions, deletions, and bitflips necessary to transform one bitstring into the other. Again, these enumerations are not cycles in  $Q_{n,[k,l]}$  and they are not tight. However, the corresponding generation algorithms are very simple and fast (loopless). Improving on this, as a byproduct of the results mentioned in the previous section we obtain a simple loopless algorithm to enumerate all vertices of  $Q_{n,[k,l]}$  (for an arbitrary interval  $[k,l]$ ) such that any two consecutive bitstrings have Hamming distance (and Levenshtein distance) at most 2.

### 1.3 Outline of this paper

In this extended abstract we restrict ourselves to only explaining the main ideas behind our constructions, algorithms and proofs. Lengthy pseudocode and proofs of auxiliary statements are omitted. Specifically, in Sections 2 and 3 we present our constructions for proving Theorems 7 and 8, respectively. In Section 4 we present our algorithm for part (a) of Theorem 9. The algorithm for part (b) and the proof of Theorem 10 as well as other omitted proofs can be found in the full version [14].

## 2 Saturating cycles

### 2.1 Trimming Gray codes and proof of Theorem 7 (i)+(ii)

In this section we sketch how to prove cases (i) and (ii) from Theorem 7 by showing that the standard reflected Gray code in  $Q_n$  mentioned in the introduction (see [13] and [21, Section 7.2.1.1]) can be ‘trimmed’ to any number of consecutive levels of  $Q_n$  so that it visits all these vertices except possibly some vertices from the first and last levels. This technique is a generalization of the approach presented in [35] to prove Theorem 6, and it yields the following result:

► **Theorem 11.** *For any  $n \geq 3$  and  $k, l$  with  $0 \leq k < l \leq n$  and  $l - k \geq 2$ , the graph  $Q_{n,[k,l]}$  has a cycle that visits all vertices except possibly some vertices from levels  $k$  and  $l$ .*

Note that if  $l - k$  is even, then the first and last level of  $Q_{n,[k,l]}$  are from the same bipartite class, so the cycle obtained from Theorem 11 is saturating, which immediately yields Theorem 7 (ii).

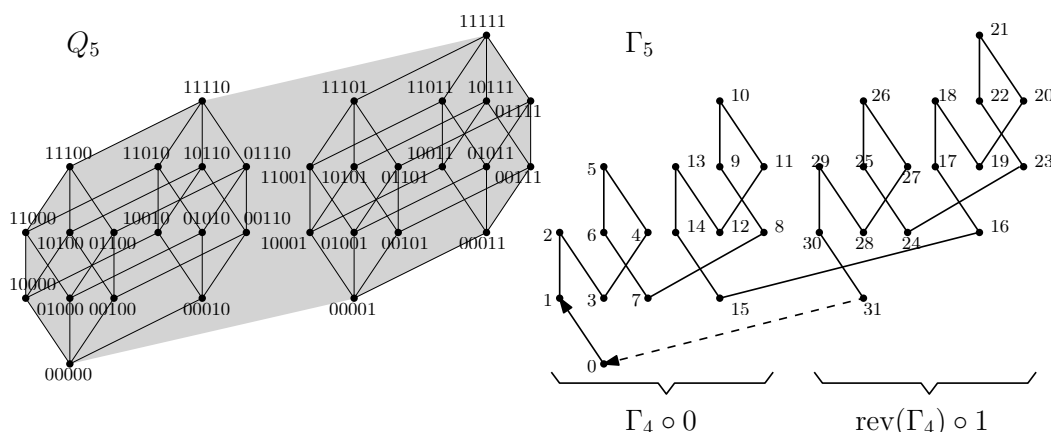
The ( $n$ -dimensional) reflected Gray code  $\Gamma_n$  is a (cyclic) sequence  $\Gamma_n$  of all vertices of  $Q_n$  defined recursively by

$$\Gamma_1 = (0, 1) , \tag{1a}$$

$$\Gamma_{n+1} = (\Gamma_n \circ 0, \text{rev}(\Gamma_n) \circ 1) , \quad n \geq 1 , \tag{1b}$$

where  $\Gamma_n \circ 0$  and  $\Gamma_n \circ 1$  denote the sequences obtained from  $\Gamma_n$  by attaching a 0-bit or 1-bit to every entry of  $\Gamma_n$ , respectively, and  $\text{rev}(\Gamma_n)$  denotes the reversed sequence. See Figure 2 for an illustration.





■ **Figure 2** The hypercube  $Q_5$  (left), where the grey area represents all 16 edges along which the last bit is flipped, and the reflected Gray code  $\Gamma_5$  in  $Q_5$  (right), where the numbers are indices of vertices in  $\Gamma_5$  (starting from 0). The dashed edge represents the adjacency between the last and first vertex of  $\Gamma_5$ .

The reflected Gray code  $\Gamma_n$  is the standard example how to enumerate all bitstrings of length  $n$  such that any two consecutive bitstrings differ in exactly one bit. For an explicit definition of  $\Gamma_n$  and further interesting properties see [21, Section 7.2.1.1].

For any  $0 \leq k \leq n$  let  $\Gamma_{n,k}$  be the subsequence of  $\Gamma_n$  that contains all vertices in level  $k$ . Moreover, we let  $s_{\Gamma_{n,k}}(x) = s(x)$  denote the successor of  $x$  in  $\Gamma_{n,k}$  (if  $x$  is the last vertex of  $\Gamma_{n,k}$ , then  $s(x)$  is the first vertex of  $\Gamma_{n,k}$ ). Similarly, let  $s_{\Gamma_n}(x)$  denote the successor of  $x$  in  $\Gamma_n$ .

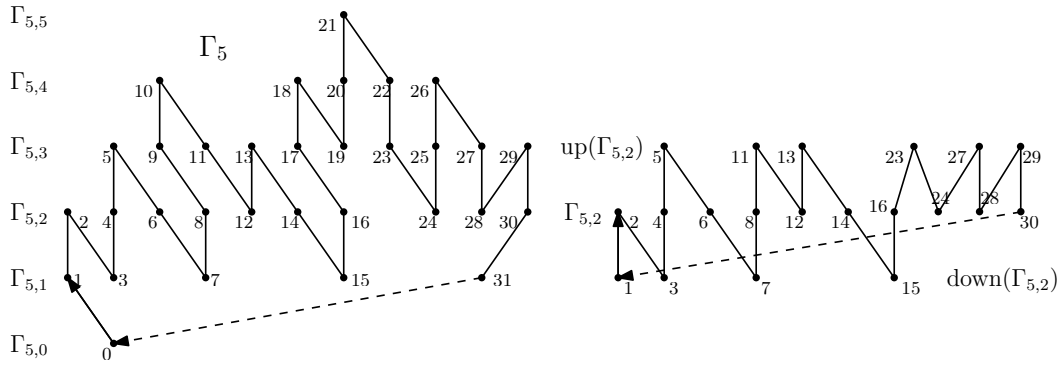
As already observed in [35], any two consecutive vertices in  $\Gamma_{n,k}$  differ in exactly two positions. The sequence  $\Gamma_{n,k}$  therefore provides an enumeration of all  $k$ -element subsets of  $[n]$  such that any two consecutive  $k$ -sets differ in exchanging a single element (recall Theorem 6).

► **Lemma 12** ([35]). *For any  $n \geq 2$  and  $0 < k < n$  let  $x$  be a vertex of  $\Gamma_{n,k}$  and  $y := s_{\Gamma_{n,k}}(x)$ . Then  $x$  and  $y$  differ in exactly 2 bits.*

Clearly, any two vertices  $x, y$  in level  $k$  at distance 2 have a unique common neighbor in level  $k - 1$  and a unique common neighbor in level  $k + 1$ , let us denote them by  $\text{down}(x, y)$  and  $\text{up}(x, y)$ , respectively. The key idea in trimming the reflected Gray code to a given sequence of consecutive levels  $[k, l]$ , where  $l - k \geq 2$ , is to replace the subpath  $P$  of  $\Gamma_n$  in  $Q_n$  between a vertex  $x$  in level  $l - 1$  and its consecutive vertex  $s_{\Gamma_{n,l-1}}(x)$  by the path  $(x, \text{up}(x, s(x)), s(x))$  if  $P$  ascends above level  $l - 1$ , and between a vertex  $x$  in level  $k + 1$  and its consecutive vertex  $s_{\Gamma_{n,k+1}}(x)$  by the path  $(x, \text{down}(x, s(x)), s(x))$  if  $P$  descends below level  $k + 1$ . See Figure 3 for an illustration.

Formally, we say that a vertex  $x$  of  $Q_n$  in level  $k$  is an *upward vertex* if  $s_{\Gamma_n}(x)$  is in level  $k + 1$ , and a *downward vertex* if  $s_{\Gamma_n}(x)$  is in level  $k - 1$  (no other case is possible). Thus, the reflected Gray code  $\Gamma_n$  ascends from upward vertices and descends from downward vertices. For any  $0 < k \leq n$ , we let  $\text{up}(\Gamma_{n,k})$  denote the sequence of all vertices  $\text{up}(x, s(x))$  in level  $k + 1$ , where  $x$  is an upward vertex of  $\Gamma_{n,k}$ , in the order induced by  $\Gamma_{n,k}$ . Similarly, for any  $0 \leq k < n$  we let  $\text{down}(\Gamma_{n,k})$  denote the sequence of all vertices  $\text{down}(x, s(x))$  in level  $k - 1$ , where  $x$  is a downward vertex of  $\Gamma_{n,k}$ , in the order induced by  $\Gamma_{n,k}$ .

The next lemma captures the key property guaranteeing that in trimming the reflected Gray code as described above we will never visit the same vertex twice. Note that



■ **Figure 3** Schematic drawing of  $\Gamma_5$  (left) highlighting the order in which levels are visited, and the corresponding sequences  $\Gamma_{5,k}$ ,  $0 \leq k \leq 5$ . See Figure 2 what the actual vertices are. The sequences  $\text{up}(\Gamma_{5,2})$ ,  $\text{down}(\Gamma_{5,2})$ , and  $\Gamma_5$  trimmed to levels 1 up to 3 of  $Q_5$  (right).

by a cyclic rotation of a sequence  $(u_1, \dots, u_{m-1}, u_m)$  to the right we mean the sequence  $(u_m, u_1, \dots, u_{m-1})$ .

► **Lemma 13.** *For any  $n \geq 2$  and  $0 < k < n$ ,  $\text{up}(\Gamma_{n,k})$  is a subsequence of  $\Gamma_{n,k+1}$ , and cyclically rotating  $\text{down}(\Gamma_{n,k})$  once to the right yields a subsequence of  $\Gamma_{n,k-1}$ .*

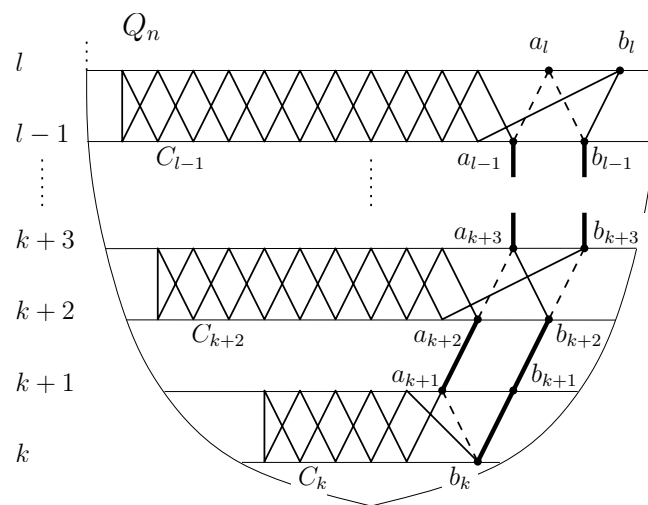
With Lemma 13 in hand we are ready to prove Theorem 11.

**Proof of Theorem 11.** We build the desired cycle by trimming the reflected Gray code  $\Gamma_n$  to levels  $[k, l]$ . Subpaths of  $\Gamma_n$  within the levels  $[k + 1, l - 1]$  remain unchanged (including the orientation). Each subpath  $P$  of  $\Gamma_n$  that descends from some downward vertex  $x$  at level  $k + 1$  to lower levels returns back to level  $k + 1$  at the vertex  $y := s_{\Gamma_{n,k+1}}(x)$ . As  $x$  and  $y$  differ in exactly 2 bits by Lemma 12, we may replace  $P$  by the path  $P' = (x, \text{down}(x, y), y)$ . Note that  $P'$  has the same end vertices and orientation as  $P$ , and visits only a single vertex at level  $k$ . After trimming all these descending paths, we visit on level  $k$  precisely the vertices of  $\text{down}(\Gamma_{n,k+1})$ . Since  $\text{down}(\Gamma_{n,k+1})$  (after one rotation to the right) is a subsequence of  $\Gamma_{n,k}$  by Lemma 13, all these vertices are distinct, and hence distinct trimmed paths may have at most end vertices in level  $k$  in common.

Similar arguments apply for trimming subpaths of  $\Gamma_n$  ascending from upward vertices at level  $l - 1$  to levels above. In this case the trimmed subpaths visit at level  $l$  precisely the vertices of  $\text{up}(\Gamma_{n,l-1})$ , and since this is a subsequence of  $\Gamma_{n,l}$  by Lemma 13, all these vertices are distinct. Therefore trimming correctly produces a cycle visiting all vertices in levels  $[k, l]$  except the vertices from level  $k$  that are not in  $\text{down}(\Gamma_{n,k+1})$  and the vertices from level  $l$  that are not in  $\text{up}(\Gamma_{n,l-1})$ . ◀

**Proof of Theorem 7 (ii).** Follows immediately from Theorem 11, using that if  $l - k$  is even, then the first and last level of  $Q_{n,[k,l]}$  are from the same bipartite class. ◀

**Proof of Theorem 7 (i).** We only consider the case  $0 = k < l \leq n$ , the other case follows by symmetry, using that  $Q_{n,[k,l]}$  is isomorphic to  $Q_{n,[n-l,n-k]}$ . The proof proceeds very similarly to the proof of Theorem 11, but we only trim the subpaths of  $\Gamma_n$  ascending above level  $l - 1$ , so that the highest level where vertices are visited is level  $l$  (no trimming is applied at the bottom level 0). We therefore obtain a cycle that visits all vertices in levels  $[0, l]$ , except the vertices from level  $l$  that are not in  $\text{up}(\Gamma_{n,l-1})$ . As the cycle omits only vertices from level  $l$ , it must be saturating. ◀



■ **Figure 4** Notations used in the proof of Theorem 7 (iii). The removed edges are dashed, the added edges are bold.

## 2.2 Gluing saturating cycles and proof sketch of Theorem 7 (iii)+(iv)

Trimming the reflected Gray code  $\Gamma_n$  to levels  $[k, l]$  as described in the last section does not yield a saturating cycle when  $l - k \geq 3$  is odd unless  $k = 0$  or  $l = n$ . In general the trimmed cycle omits some vertices from both levels  $k$  and  $l$ , which are from different bipartite classes for odd  $l - k$ . We therefore use a different strategy to prove Theorem 7 (iii): We glue together several saturating cycles obtained from Theorem 3 (see Figure 4 for an illustration). To be able to join the cycles across different levels, each cycle between levels  $i$  and  $i + 1$  is first transformed by applying a suitable bit permutation (an automorphism of  $Q_{n,[i,i+1]}$ ), in such a way that the permuted cycle visits certain distinguished vertices  $a_i$  and  $b_i$  in levels  $i$  and  $i + 1$  in a certain order, and so that it omits certain other vertices. The cycles are then glued together by removing one or two edges from each cycle and by joining the resulting paths by adding a few other cube edges. The details of this construction, in particular the definition of the vertices  $a_i$  and  $b_i$ , can be found in the full version [14]. This gluing approach yields a saturating cycle only if all involved levels are either below or above the middle (this is reflected by the conditions  $l \leq \lceil n/2 \rceil$  or  $\lfloor n/2 \rfloor \leq k$  in Theorem 7 (iii)). Otherwise the omitted vertices would be from different bipartite classes, so the resulting cycle would not be saturating. To prove Theorem 7 (iv), we inductively glue together pairs of saturating cycles of smaller dimension. Both proofs are similar to the approach presented in [26].

## 3 Tight enumerations

We call a sequence  $C$  that contains each vertex of  $Q_{n,[k,l]}$  exactly once an *enumeration of the vertices of  $Q_{n,[k,l]}$*  (recall that the successor of the last vertex of  $C$  equals the first vertex of  $C$ ). The *total distance* of the enumeration  $C$  is  $\text{td}(C) := \sum_{u \in C} d(u, s_C(u))$ , where  $d(x, y)$  denotes the Hamming distance between  $x$  and  $y$ . As any two consecutive bitstrings in any enumeration have distance at least 1 and distance at least 2 if they are from the same bipartite class of  $Q_{n,[k,l]}$ , we have

$$\text{td}(C) \geq v(Q_{n,[k,l]}) + \delta(Q_{n,[k,l]}) \tag{2}$$

(both the number of vertices  $v(Q_{n,[k,l]})$  and the difference between the larger and smaller partition class  $\delta(Q_{n,[k,l]})$  can be easily computed explicitly). A *tight enumeration* is an enumeration for which the lower bound (2) is attained. Clearly, an enumeration is tight, if and only if it has only distance-1 and distance-2 steps, and all the distance-2 steps are within the same partition class of  $Q_{n,[k,l]}$  (this will be the larger partition class, and there will be exactly  $\delta(Q_{n,[k,l]})$  such distance-2 steps in this class).

The proof of Theorem 8 is very similar to the proof of Theorem 7, and the key ideas in the different cases (i)–(iv) are the same. In this extended abstract, we only present the proofs for cases (i)–(ii). The proofs for cases (iii)–(iv) can be found in [14].

### 3.1 Proof of Theorem 8 (i)–(ii)

For  $k = l$  we have  $v(Q_{n,[k,k]}) = \delta(Q_{n,[k,k]}) = \binom{n}{k}$  and a tight enumeration of all weight  $k$  bitstrings of length  $n$  is given by  $\Gamma_{n,k}$ , by Lemma 12 (this is exactly the proof of Theorem 6 presented in [35]). We now proceed to prove cases (i)–(ii) of Theorem 8.

**Proof of Theorem 8 (ii).** We trim the reflected Gray code  $\Gamma_n$  to levels  $[k, l]$ , but in a slightly different fashion from the proof of Theorem 11. Specifically, subpaths of  $\Gamma_n$  within the levels  $[k, l]$  remain unchanged (including the orientation). Moreover, each subpath  $P$  of  $\Gamma_n$  that descends from a downward vertex  $x$  in level  $k$  returns back to level  $k$  at the vertex  $y := s_{\Gamma_{n,k}}(x)$ , and we replace  $P$  by the distance-2 step  $(x, y)$  (recall Lemma 12). Similarly, each subpath  $P$  of  $\Gamma_n$  that ascends from an upward vertex  $x$  in level  $l$  returns back to level  $l$  at the vertex  $y := s_{\Gamma_{n,l}}(x)$ , and we replace  $P$  by the distance-2 step  $(x, y)$ . This yields an enumeration  $C$  of all vertices of  $Q_{n,[k,l]}$ . Moreover, since  $l - k$  is even, levels  $k$  and  $l$  of  $Q_{n,[k,l]}$  belong to the same bipartite class, so all distance-2 steps of  $C$  are within the same bipartite class. It follows that  $C$  is a tight enumeration. ◀

**Proof of Theorem 8 (i).** We only consider the case  $0 = k < l \leq n$ , the other case follows by symmetry. The proof proceeds very similarly as the proof of part (ii), but we only trim the subpaths of  $\Gamma_n$  ascending above level  $l$  (no trimming is applied at the bottom level 0). This yields an enumeration  $C$  of all vertices of  $Q_{n,[0,l]}$ . Moreover, all distance-2 steps of  $C$  are within the same bipartite class (in level  $l$ ), implying that  $C$  is a tight enumeration. ◀

## 4 A loopless algorithm for trimmed Gray codes

In this section we present a loopless algorithm to generate trimmed Gray codes, which is needed to prove part (a) of Theorem 9 (algorithms for cases (i) and (ii) of Theorems 7 and 8). The correctness proof and the runtime analysis for this algorithm can be found in the full version [14]. Also, the description of an efficient algorithm to compute glued Gray codes, which is needed to prove part (b) of Theorem 9 (algorithms for case (iii) of Theorems 7 and 8), can be found in [14].

Loopless algorithms both for the reflected Gray code  $\Gamma_n$  and for its restriction  $\Gamma_{n,k}$  to one level of the cube (i.e, an algorithmic version of Theorem 6) were already provided in [2, 10]. However, these two algorithms cannot simply be merged into a loopless algorithm producing the trimmed Gray code. Instead, we provide a loopless algorithm that is based on an explicit description of successor vertices. This description is a simple (constant-time computable) rule describing which bit positions of a given vertex  $x$  from  $\Gamma_{n,k}$  have to be flipped to reach the next vertex  $s_{\Gamma_{n,k}}(x)$ . In this extended abstract we do not explicitly state these rules, they are however used implicitly in the following pseudocode.

**Algorithm 1:** TRIMGC( $n, k, l$ )**Input:** Integers  $n \geq 2$  and  $0 \leq k < l \leq n$ ,  $l - k \geq 2$ .**Result:** The algorithm visits all vertices of the trimmed Gray code in  $Q_{n,[k,l]}$  (which is a saturating cycle in cases (i) and (ii) of Theorem 7).

---

```

T1  $c := k + 1; x := 1^c 0^{n-c};$  /* initialize current level  $c$ , vertex  $x \dots$  */
T2  $\nu_c := 1;$  for  $i := 1$  to  $c$  do  $p_i := c - i + 1$  /* ... and  $(\nu_1, \dots, \nu_c), (p_1, \dots, p_c)$  */
T3 while not enough vertices visited do
T4   if  $(c \text{ is even} \wedge x_1 = 0) \vee (c \text{ is odd} \wedge p_c < n \wedge x_{p_c+1} = 0)$  then  $\text{up} := \text{true}$  else  $\text{up} := \text{false}$ 
T5   if  $(\text{up} = \text{true} \wedge c < l - 1)$  then /* follow  $\Gamma_n$  up */
T6     if  $c$  is even then /*  $x_1 = 0$  */
T7        $x_1 := 1; \text{VISIT}(x); c := c + 1; p_c := 1$ 
T8       if  $x_2 = 0$  then  $\nu_c := c$  else  $\nu_c := \nu_{c-1}$ 
T9     else /*  $c$  is odd and  $x_{i+1} = 0$  */
T10       $i := p_c; x_{i+1} := 1; \text{VISIT}(x); c := c + 1; p_c := i; p_{c-1} := i + 1$ 
T11      if  $(i + 1 = n \vee x_{i+2} = 0)$  then  $\nu_c := c - 1$  else  $\nu_c := \nu_{c-2}$ 
T12   else if  $(\text{up} = \text{false} \wedge c > k + 1)$  then /* follow  $\Gamma_n$  down */
T13     if  $c$  is even then /*  $x_1 = 1$  */
T14        $x_1 := 0; \text{VISIT}(x); c := c - 1$ 
T15       if  $x_2 = 1$  then  $\nu_c := \nu_{c+1}$ 
T16     else /*  $c$  is odd and  $x_{i+1} = 1$  */
T17        $i := p_c; x_{i+1} := 0; \text{VISIT}(x); c := c - 1; p_c := i; \nu_c := c$ 
T18       if  $x_{i+2} = 1$  then  $\nu_{c-1} := \nu_{c+1}$ 
T19   else if  $(\text{up} = \text{true} \wedge c = l - 1)$  then /* follow  $\Gamma_{n,c}$  through  $\text{up}(x, s_{\Gamma_{n,c}}(x))$  */
T20     if  $c$  is even then /*  $x_{i-1} = 0$  and  $x_i = 1$  */
T21        $i := p_c; x_{i-1} := 1; \text{VISIT}(x); x_i := 0; \text{VISIT}(x); p_c := i - 1$ 
T22       if  $x_{i+1} = 1$  then  $\nu_{c-1} := \nu_c; \nu_c := c$ 
T23     else /*  $c$  is odd,  $x_{i+1} = 0$  and  $x_i = 1$  */
T24        $i := p_c; x_{i+1} := 1; \text{VISIT}(x); x_i := 0; \text{VISIT}(x); p_c := i + 1$ 
T25       if  $(i + 2 \leq n \wedge x_{i+2} = 1)$  then  $\nu_c := \nu_{c-1}$ 
T26   else /*  $\text{up} = \text{false} \wedge c = k + 1$ ; follow  $\Gamma_{n,c}$  through  $\text{down}(x, s_{\Gamma_{n,c}}(x))$  */
T27     if  $x_1 = 0$  then  $i := 1$  else  $i := p_{\nu_c} + 1$  /*  $i$  is minimal s.t.  $x_i = 0$  */
T28     if  $c \not\equiv i \pmod{2}$  then /*  $x_{i-2} = 1$  and  $x_i = 0$  */
T29        $x_{i-2} := 0; \text{VISIT}(x); x_i := 1; \text{VISIT}(x); p_{\nu_c+1} := i - 1; p_{\nu_c} := i$ 
T30       if  $(i + 1 \leq n \wedge x_{i+1} = 1)$  then  $\nu_{\nu_c+1} := \nu_{\nu_c-1}$  else  $\nu_{\nu_c+1} := \nu_c$ 
T31       if  $i > 3$  then  $\nu_c := \nu_c + 2$ 
T32     else /*  $c \equiv i \pmod{2}; j > i$  is minimal s.t.  $x_j = 1$  */
T33       if  $x_1 = 0$  then  $a := c; j := p_a$  else  $a := \nu_c - 1; j := p_a$ 
T34       if  $j = n$  then  $x_n := 0; \text{VISIT}(x); x_i := 1; \text{VISIT}(x); p_1 := i; \nu_c := 1$ 
T35       else if  $x_{j+1} = 0$  then /*  $j < n, x_{i-1} = 1$  and  $x_{j+1} = 0$  */
T36          $x_{i-1} := 0; \text{VISIT}(x); x_{j+1} := 1; \text{VISIT}(x); p_{\nu_c} := j; p_{\nu_c-1} := j + 1$ 
T37         if  $(j + 2 \leq n \wedge x_{j+2} = 1)$  then  $\nu_{\nu_c} := \nu_{\nu_c-2}$  else  $\nu_{\nu_c} := \nu_c - 1$ 
T38         if  $i > 2$  then  $\nu_c := \nu_c + 1$ 
T39       else /*  $j < n, x_{j+1} = 1$  and  $x_i = 0$  */
T40          $x_{j+1} := 0; \text{VISIT}(x); x_i := 1; \text{VISIT}(x); p_a := i; p_{a-1} := j$ 
T41         if  $(j + 2 \leq n \wedge x_{j+2} = 1)$  then  $\nu_{a-2} := \nu_a$ 
T42         if  $j = i + 1$  then  $\nu_c := a - 1$  else  $\nu_{a-1} := a - 1, \nu_c := a$ 

```

---

Consider now the algorithm  $\text{TRIMGC}(n, k, l)$  that computes a trimmed Gray code between levels  $k$  and  $l$  with  $l - k \geq 2$  of  $Q_n$  as described in Section 2.1 (Theorem 11), i.e., the algorithm produces a saturating cycle for cases (i) and (ii) of Theorem 7. At the end of this section we describe how to modify the algorithm to produce a tight enumeration for cases (i) and (ii) of Theorem 8.

The algorithm maintains the current vertex in the variable  $x$  and the current level in the variable  $c$ ,  $k < c < l$ . Both are initialized in line T1 to  $c = k + 1$  and  $x = 1^c 0^{n-c}$ , where  $b^k$  denotes the  $k$ -fold repetition of the symbol  $b \in \{0, 1\}$  (the code can easily be modified to start at a different vertex). The algorithm visits the sequence of vertices along the trimmed Gray code by the calls  $\text{VISIT}(x)$ , which could be some user-defined function that reads  $x$ . For simplicity the main while-loop does not have an explicit termination criterion, but this can be added easily (e.g., by providing an additional argument to the algorithm that specifies the number of vertices to be visited before termination). If the while-loop is not terminated, then the same cycle is traversed over and over again. There are four main cases distinguished in the while-loop of the algorithm: If the current level is between  $k + 1$  and  $l - 1$ , then we either follow  $\Gamma_n$  by flipping a 0-bit to a 1-bit (if the condition in line T5 is satisfied), or we follow  $\Gamma_n$  by flipping a 1-bit to a 0-bit (if the condition in line T12 is satisfied). If the current level is  $c = l - 1$  and  $x$  is an upward vertex (the condition in line T19 is satisfied), then we first flip a 0-bit to the vertex  $\text{up}(x, y)$ ,  $y := s_{\Gamma_{n,c}}(x)$ , and then a 1-bit to the vertex  $y$ . On the other hand, if the current level is  $c = k + 1$  and  $x$  is a downward vertex (the condition in line T26 is satisfied), then we first flip a 1-bit to the vertex  $\text{down}(x, y)$ ,  $y := s_{\Gamma_{n,c}}(x)$ , and then a 0-bit to the vertex  $y$ .

The key to our loopless algorithm is to be able to determine in constant time the smallest integer  $i \geq 1$  with  $x_i = 1$  or with  $x_i = 0$  (this part of the before-mentioned successor rule computation). Furthermore, we also need the smallest integer  $j > i$  with  $x_j = 1$ . To achieve this the algorithm maintains the following data structures: We maintain an array  $(p_1, p_2, \dots, p_c)$  with the positions of the 1-bits in  $x = (x_1, x_2, \dots, x_n)$  where  $p_i$ ,  $1 \leq i \leq c$ , is the position of the  $i$ -th 1-bit in  $x$  counted from the right (from the highest index  $n$ ). Thus  $p_c$  is the position of the first 1-bit in  $x$  from the left (from the lowest index 1). It turns out that adding and removing 1's happens around the position  $p_c$ , so the length of this array changes dynamically. For  $i > c$  the value of  $p_i$  is undefined (the algorithm does not ‘clean up’ those values after using them). We also maintain an array  $(\nu_1, \nu_2, \dots, \nu_c)$  to quickly find the smallest integer  $j > i$  with  $x_j = 0$  where  $i$  is the smallest integer with  $x_i = 1$ . However, the value of  $\nu_i$ ,  $1 \leq i \leq c$ , is defined only if  $p_i$  is a starting position of a substring of 1-bits in  $x$ ; i.e.,  $x_{p_i} = 1$  and  $x_{p_i-1} = 0$ , or  $p_i = 1$ . In this case, the value of  $\nu_i$  is the index such that  $p_{\nu_i}$  is the position where the corresponding substring of 1-bits ends in  $x$ . In particular, since  $p_c$  is the position of the first 1 in  $x$ , the value  $p_{\nu_c} + 1$  is the smallest integer greater than  $p_c$  such that  $x$  has a 0-bit at this position. Initially, there is only one substring of 1-bits in  $x = 1^c 0^{n-c}$  that starts at position  $p_c = 1$  and ends at position  $p_1 = c$ , so we have  $\nu_c = 1$  (see line T2). Assuming that the algorithm correctly computes the positions to flip the next bit(s) of  $x$ , it can be verified straightforwardly that it also correctly updates all relevant entries of  $p$  and  $\nu$  in the four main cases (and their subcases).

To obtain a loopless algorithm that generates a tight enumeration of the vertices of  $Q_{n,[k,l]}$  (instead of a saturating cycle), we simply call  $\text{TRIMGC}(n, k - 1, l + 1)$  and omit the first of the two  $\text{VISIT}(x)$  calls in each of the lines T21, T24, T29, T34, T36 and T40. The algorithm then moves directly with a distance-2 step from a vertex  $x$  in level  $k$  or  $l$  to the vertex  $s_{\Gamma_{n,k}}(x)$  or  $s_{\Gamma_{n,l}}(x)$ , respectively, without visiting  $\text{down}(x, s_{\Gamma_{n,k}}(x))$  or  $\text{up}(x, s_{\Gamma_{n,l}}(x))$  in between. Also, if  $k = 0$  then line T2 has to be omitted, and the special case  $c = 1$  and

$x = 0^{n-1}1$  must be treated separately in lines T17 and T18 (by setting  $i := n - 1$ ). This completes the description of the algorithms for part (a) of Theorem 9. As mentioned before, the missing correctness proofs and the runtime analysis can be found in [14].

**Acknowledgements.** The authors thank Jiří Fink, Jerri Nummenpalo and Robert Šámal for several stimulating discussions about the problems discussed in this paper and the anonymous reviewers for their helpful comments.

---

## References

- 1 URL: <http://www.math.tu-berlin.de/~muetze>.
- 2 J. Bitner, G. Ehrlich, and E. Reingold. Efficient generation of the binary reflected Gray code and its applications. *Comm. ACM*, 19(9):517–521, 1976.
- 3 M. Buck and D. Wiedemann. Gray codes with restricted density. *Discrete Math.*, 48(2-3):163–171, 1984. doi:10.1016/0012-365X(84)90179-1.
- 4 P. Chase. Combination generation and graylex ordering. *Congr. Numer.*, 69:215–242, 1989. Eighteenth Manitoba Conference on Numerical Mathematics and Computing (Winnipeg, MB, 1988).
- 5 P. Diaconis and R. Graham. *Magical mathematics*. Princeton University Press, Princeton, NJ, 2012. The mathematical ideas that animate great magic tricks, With a foreword by Martin Gardner.
- 6 D. Duffus, H. Kierstead, and H. Snevily. An explicit 1-factorization in the middle of the Boolean lattice. *J. Combin. Theory Ser. A*, 65(2):334–342, 1994. doi:10.1016/0097-3165(94)90030-2.
- 7 D. Duffus, B. Sands, and R. Woodrow. Lexicographic matchings cannot form Hamiltonian cycles. *Order*, 5(2):149–161, 1988. doi:10.1007/BF00337620.
- 8 P. Eades, M. Hickey, and R. Read. Some Hamilton paths and a minimal change algorithm. *J. Assoc. Comput. Mach.*, 31(1):19–29, 1984. doi:10.1145/2422.322413.
- 9 P. Eades and B. McKay. An algorithm for generating subsets of fixed size with a strong minimal change property. *Inform. Process. Lett.*, 19(3):131–133, 1984. doi:10.1016/0020-0190(84)90091-7.
- 10 G. Ehrlich. Loopless algorithms for generating permutations, combinations, and other combinatorial configurations. *J. Assoc. Comput. Mach.*, 20:500–513, 1973.
- 11 M. El-Hashash and A. Hassan. On the Hamiltonicity of two subgraphs of the hypercube. In *Proceedings of the Thirty-second Southeastern International Conference on Combinatorics, Graph Theory and Computing (Baton Rouge, LA, 2001)*, volume 148, pages 7–32, 2001.
- 12 S. Felsner and W. Trotter. Colorings of diagrams of interval orders and  $\alpha$ -sequences of sets. *Discrete Math.*, 144(1-3):23–31, 1995. *Combinatorics of ordered sets (Oberwolfach, 1991)*. doi:10.1016/0012-365X(94)00283-0.
- 13 F. Gray. Pulse code communication. March 17, 1953 (filed Nov. 1947). U.S. Patent 2,632,058.
- 14 P. Gregor and T. Mütze. Trimming and gluing Gray codes. *arXiv:1607.08806*, full version of this extended abstract, Jan 2017. URL: <https://arxiv.org/abs/1607.08806>.
- 15 P. Gregor and R. Škrekovski. On generalized middle-level problem. *Inform. Sci.*, 180(12):2448–2457, 2010. doi:10.1016/j.ins.2010.02.009.
- 16 I. Havel. Semipaths in directed cubes. In *Graphs and other combinatorial topics (Prague, 1982)*, volume 59 of *Teubner-Texte Math.*, pages 101–108. Teubner, Leipzig, 1983.
- 17 P. Horák, T. Kaiser, M. Rosenfeld, and Z. Ryjáček. The prism over the middle-levels graph is Hamiltonian. *Order*, 22(1):73–81, 2005. doi:10.1007/s11083-005-9008-7.

- 18 T. Jenkyns and D. McCarthy. Generating all  $k$ -subsets of  $\{1 \cdots n\}$  with minimal changes. *Ars Combin.*, 40:153–159, 1995.
- 19 R. Johnson. Long cycles in the middle two layers of the discrete cube. *J. Combin. Theory Ser. A*, 105(2):255–271, 2004. doi:10.1016/j.jcta.2003.11.004.
- 20 H. Kierstead and W. Trotter. Explicit matchings in the middle levels of the Boolean lattice. *Order*, 5(2):163–171, 1988. doi:10.1007/BF00337621.
- 21 D. Knuth. *The Art of Computer Programming, Volume 4A*. Addison-Wesley, 2011.
- 22 S. Locke and R. Stong. Problem 10892: Spanning cycles in hypercubes. *Amer. Math. Monthly*, 110:440–441, 2003.
- 23 T. Mütze. Proof of the middle levels conjecture. *Proc. Lond. Math. Soc.*, 112(4):677–713, 2016. doi:10.1112/plms/pdw004.
- 24 T. Mütze and J. Nummenpalo. Efficient computation of middle levels Gray codes. In N. Bansal and I. Finocchi, editors, *Algorithms – ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 915–927. Springer Berlin Heidelberg, 2015. arXiv:1506.07898. doi:10.1007/978-3-662-48350-3\_76.
- 25 T. Mütze and J. Nummenpalo. A constant-time algorithm for middle levels Gray codes. arXiv:1606.06172. An extended abstract has been accepted for presentation at SODA 2017, June 2016.
- 26 T. Mütze and P. Su. Bipartite Kneser graphs are Hamiltonian. In J. Nešetřil, O. Serra, and J. A. Telle, editors, *Electronic Notes in Discrete Mathematics – Eurocomb 2015*, volume 49, pages 259–267. Elsevier, 2015. arXiv:1503.09175. Accepted for publication in *Combinatorica*.
- 27 T. Mütze and F. Weber. Construction of 2-factors in the middle layer of the discrete cube. *J. Combin. Theory Ser. A*, 119(8):1832–1855, 2012. doi:10.1016/j.jcta.2012.06.005.
- 28 F. Ruskey. Adjacent interchange generation of combinations. *J. Algorithms*, 9(2):162–180, 1988. doi:10.1016/0196-6774(88)90036-3.
- 29 C. Savage. Long cycles in the middle two levels of the Boolean lattice. *Ars Combin.*, 35-A:97–108, 1993.
- 30 C. Savage. A survey of combinatorial Gray codes. *SIAM Rev.*, 39(4):605–629, 1997. doi:10.1137/S0036144595295272.
- 31 C. Savage and P. Winkler. Monotone Gray codes and the middle levels problem. *J. Combin. Theory Ser. A*, 70(2):230–248, 1995. doi:10.1016/0097-3165(95)90091-8.
- 32 I. Shields, B. Shields, and C. Savage. An update on the middle levels problem. *Discrete Math.*, 309(17):5271–5277, 2009. doi:10.1016/j.disc.2007.11.010.
- 33 M. Shimada and K. Amano. A note on the middle levels conjecture. arXiv:0912.4564, Sep 2011.
- 34 B. Stevens and A. Williams. The coolest way to generate binary strings. *Theory Comput. Syst.*, 54(4):551–577, 2014. doi:10.1007/s00224-013-9486-8.
- 35 D. Tang and C. Liu. Distance-2 cyclic chaining of constant-weight codes. *IEEE Trans. Computers*, C-22:176–180, 1973.
- 36 P. Winkler. *Mathematical puzzles: a connoisseur’s collection*. A K Peters, Ltd., Natick, MA, 2004.



# Mixing of Permutations by Biased Transposition\*

Shahrzad Haddadan<sup>1</sup> and Peter Winkler<sup>2</sup>

1 Sapienza University of Rome, Dipartimento di Informatica, Rome, Italy  
shahrzad.haddadan@gmail.com

2 Department of Mathematics, Dartmouth College, Hanover, NH, USA  
peter.winkler@Dartmouth.edu

---

## Abstract

Markov chains defined on the set of permutations of  $1, 2, \dots, n$  have been studied widely by mathematicians and theoretical computer scientists [15, 4, 1]. We consider chains in which a position  $i < n$  is chosen uniformly at random, and then  $\sigma(i)$  and  $\sigma(i+1)$  are swapped with probability depending on  $\sigma(i)$  and  $\sigma(i+1)$ . Our objective is to identify some conditions that assure rapid mixing.

One case of particular interest is what we call the “gladiator chain,” in which each number  $g$  is assigned a “strength”  $s_g$  and when  $g$  and  $g'$  are swapped,  $g$  comes out on top with probability  $s_g/(s_g + s_{g'})$ . The stationary probability of this chain is the same as that of the slow-mixing “move ahead one” chain for self-organizing lists, but an open conjecture of Jim Fill’s implies that all gladiator chains mix rapidly. Here we obtain some positive partial results by considering cases where the gladiators fall into only a few strength classes.

**1998 ACM Subject Classification** G.3 Probability and Statistics, G.2.1 Combinatorics

**Keywords and phrases** Markov chains, permutations, self organizing lists, mixing time

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.41

## 1 Introduction

For any arbitrary natural number  $n \in \mathbb{N}$ , we define  $S_n$  to be the set that contains all the permutations of numbers  $1, 2, \dots, n$ . A natural Markov chain on  $S_n$  is the chain which picks a number  $1 \leq i \leq n-1$  uniformly at random and operating on  $\sigma \in S_n$ , puts  $\sigma(i+1)$  ahead of  $\sigma(i)$  w.p.  $p_{\sigma(i), \sigma(i+1)}$ . We call such chains *adjacent transposition* Markov chains. These chains have been studied widely for various choices of  $p_{i,j}$  [15, 5, 1, 2].

In this paper, we consider the total variation mixing time, which is defined as the time it takes until the total variation distance between the distribution of the current state and stationarity is less than  $\epsilon$  (where  $\epsilon$  is some fixed quantity in  $(0,1)$ ). For a Markov chain  $\mathcal{M}$  we denote this time by  $t_\epsilon(\mathcal{M})$ , or if  $\epsilon = 1/4$ , simply by  $t(\mathcal{M})$ .

Jim Fill<sup>1</sup> conjectured that: If the adjacent transposition Markov chain is monotone, then it is rapidly mixing (meaning the mixing time is polynomial in  $n$ ). Monotonicity in this context means that for all  $i, j$  satisfying  $1 \leq i < j \leq n$ ,  $p_{i,j} \geq 1/2$  and  $p_{i,j-1} \leq p_{i,j}$  and  $p_{i+1,j} \leq p_{i,j}$ . [5].

Here we provide a brief history of the results on adjacent transposition Markov chains. All of the chains below are monotone and rapidly mixing. Wilson and Benjamini’s papers [15, 1] led to Fill’s conjecture [5]; Bhakta et al. [2] verified the conjecture in two cases.

---

\* Research supported by NSF grant DMS-1162172.

<sup>1</sup> Fill considered the spectral gap (another measure of mixing) in his study. Here, we are interested in total variation mixing time, which in this case is within a polynomial factor of the spectral gap.



**1. The simple chain.** In the case where  $p_{i,j} = 1/2$  for all  $i$  and  $j$ , the chain will have a simple description: Given a permutation  $\sigma$ , pick two adjacent elements uniformly at random, and flip a fair coin to decide whether to swap them. We call this chain, whose stationary distribution is uniform, the *simple* chain. Ironically, proving precise mixing results for this chain was not simple. Many papers targeted this problem [4, 3], and finally Wilson [15] showed the mixing time for this chain is  $\Theta(n^3 \log n)$  (proving lower and upper bounds within constant factors).

**2. The constant-bias chain.** After Wilson's paper, Benjamini et al. [1] studied the case where  $p_{i,j} = p > 1/2$  for all  $i < j$ , and  $p_{j,i} = 1-p$ , and they showed the constant-bias chain, mixes in  $\Theta(n^2)$  steps.

**3. "Choose your weapon" and "league hierarchy" chains.** The following two special cases were studied by Bhakta et al. [2]: the *choose your weapon chain* where  $p_{i,j}$  is only dependent on  $i$ , and the *league hierarchy chain* given by a binary tree  $T$  with  $n$  leaves. Each interior node  $v$  of  $T$  is labeled with some probability  $1/2 \leq q_v \leq 1$ , and the leaves are labeled by numbers  $1 \dots n$ . The probability of putting  $j$  ahead of  $i$  for  $j > i$  is equal to  $p_{i,j} = q_{j \wedge i}$  where  $j \wedge i$  is the node that is the lowest common ancestor of  $i$  and  $j$  in  $T$ . Bhakta et al. showed that the choose your weapon chain mixes in  $\mathcal{O}(n^8 \log n)$  steps and the league hierarchy chain mixes in  $\mathcal{O}(n^4 \log n)$  steps.

Here we are interested in *gladiator Markov chains* which constitute a subclass of the monotone adjacent transposition chains. These chains have a connection to self organizing lists, and were introduced by Jim Fill.

Fill was interested in probabilistic analysis of algorithms for *self-organizing lists* (SOLs). Self-organizing lists are data structures that facilitate linear searching in a list of records; the objective of a self-organizing list is to sort the records in non-decreasing order of their access frequencies [13]. Since these frequencies are not known in advance, an SOL algorithm aims to move a particular record ahead in the list when access on that record is requested. There are two widely used SOL algorithms: the *move ahead one* algorithm (MA1) and the *move to front* algorithm (MTF). In MA1, if the current state of the list is  $(x_1, x_2, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n)$  and the  $i$ th record is requested for access, it will go ahead in the list only one position and the list will be modified to  $(x_1, x_2, \dots, x_i, x_{i-1}, x_{i+1}, \dots, x_n)$ . In MTF it will go to the front and the list will be modified to  $(x_i, x_1, x_2, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$ . It appears that MA1 should perform better than MTF when the list is almost sorted and worse when the low frequency records are standing in front; however, this has not been analytically studied [6]. Considering the adjacent transposition Markov chain corresponding to MA1, Fill shows [5] that there are cases in which the chain is not rapidly mixing. Hence, he poses the question of sampling from the stationary distribution of MA1, and he introduces the gladiator chain which has the same stationary distribution as MA1 and seems to be rapidly mixing for arbitrary choice of parameters.

In a gladiator chain, each element  $i$  can be thought of as a gladiator with strength  $s(i)$ . Every permutation of numbers  $1, 2, \dots, n$  can be thought of as a ranking of gladiators. In each step of the Markov chain we choose  $1 \leq k < n$  uniformly at random, i.e. we choose adjacent gladiators  $\sigma(k) = i$  and  $\sigma(k+1) = j$ . These gladiators will fight over their position in ranking. With probability  $p_{j,i} = s(i)/(s(i) + s(j))$ , gladiator  $i$  will be the winner of the game and will be placed ahead of  $j$  in  $\sigma$  if it isn't already. With probability  $p_{i,j} = 1 - p_{j,i}$ ,  $j$  is put ahead of  $i$ . If Fill's conjecture holds the gladiator chain must mix rapidly.

Another interesting Markov chain which has received a lot of attention is the *exclusion process* ([11, 10]). In this Markov chain we have a graph  $G = \langle V, E \rangle$  and  $m < |V|$  particles on the vertices of  $G$ . The sample space is the set containing all the different placements of the  $m$  particles on vertices of  $G$ . At each step of the Markov chain we pick a vertex  $v$  uniformly at random with probability  $1/|V|$  and one of its adjacent vertices,  $w$  with probability  $1/d(v)$ . If there is a particle in one of the vertices and not the other one, we swap the position of the particle with a constant probability  $p$ . We are interested in the exclusion process when the graph is a finite path with  $n$  vertices. We will see that this chain has connections with the gladiator chain. This chain was studied by Benjamini et al. [1] and is known to mix in  $\Theta(n^2)$  steps<sup>2</sup>.

**Our Contribution.** We study the gladiator chain when the gladiators fall into a constant number of teams, and gladiators in each team have the same strength (Definition 1). We then extend the definition of the exclusion process (studied by Benjamini et al.) by allowing particles of different types to swap their positions on a linear line. We call this new chain a *linear particle system* (Definition 2). We show that mixing results for linear particle systems can produce mixing results in gladiator Markov chains (Theorem 4).

We study the linear particle system in which there are three particle types, and in Theorem 5 we extend Benjamini et al.'s result by showing the three particle system mixes rapidly; this is our main result. Having Theorem 5 we conclude that the following adjacent transposition chains mix rapidly, and hence confirming Fill's conjecture in these cases: the gladiator chain when there are three teams of same-strength gladiators, and the league hierarchy chain for ternary trees (extending Bhakta et al.'s work [2]).

► **Remark.** We believe the linear particle systems, like the exclusion processes are interesting Markov chains that may appear as components of other Markov chains, and thus would facilitate studying mixing times of other chains (For instance see Corollary 7 in which we extend a result about binary trees to ternary trees).

Definitions and results are presented in Section 2, along with the correspondence between gladiator chains and linear particle systems. Section 3 contains the proof that the three-type system mixes rapidly under certain conditions.

## 2 Definitions and Results

► **Definition 1. Gladiator chain** (Playing in teams). Consider the Markov chain on state space  $S_n$  that has the following properties: The set  $[n]$  (i.e. gladiators) can be partitioned into subsets:  $T_1, T_2, \dots, T_k$  ( $k$  teams). We have the following strength function:  $s : [n] \rightarrow \mathbb{R}$ ,  $s(g) = s_j$  iff  $g \in T_j$ . At each step of Markov chain, we choose  $i \in [n-1]$  uniformly at random. Given that we are at state  $\sigma$ , and  $\sigma(i) = g, \sigma(i+1) = g'$ , we put  $g$  ahead of  $g'$  with probability  $\frac{s(g)}{s(g)+s(g')}$ . We denote a gladiator chain having  $n$  gladiators playing in  $k$  different teams by  $\mathcal{G}_k(n)$ .<sup>3</sup>

<sup>2</sup> Benjamini et al. use this result to prove that the constant-biased adjacent transposition chain is rapidly mixing.

<sup>3</sup> Although the notation  $\mathcal{G}_k(n_1, n_2, \dots, n_k)$  would be more precise ( $n_i$  being cardinality of  $T_i$ ), we avoid using it for simplicity and also because our analysis is not dependent on  $n_1, n_2, \dots, n_k$ .

## 41:4 Mixing of Permutations by Biased Transposition

This is a reversible Markov chain and the stationary distribution  $\pi$  is

$$\pi(\sigma) = \prod_{i=1}^n s(i)^{\sigma^{-1}(i)} / Z. \quad (Z \text{ is a normalizing factor.}) \quad (1)$$

Note that by writing  $\sigma(i) = g$  we mean gladiator  $g$  is located at position  $i$  in  $\sigma$ . By writing  $\sigma^{-1}(g)$  we are referring to the position of gladiator  $g$  in the permutation  $\sigma$ . We use this notation throughout the text and for permutations presenting both gladiators and particles.

► **Definition 2. Linear particle systems.** Assume we have  $k$  types of particles and of each type  $i$ , we have  $n_i$  indistinguishable copies. Let  $n = \sum_{i=1}^k n_i$ . Let  $\Omega_{n_1, n_2, \dots, n_k}$  be the state space containing all the different linear arrangements of these  $n$  particles. If the current state of the Markov chain is  $\sigma$ , choose  $i \in [1, n-1]$  uniformly at random. Let  $\sigma(i)$  be of type  $t$  and  $\sigma(i+1)$  be of type  $t'$ . If  $t = t'$  do nothing. Otherwise, put  $\sigma(i)$  ahead of  $\sigma(i+1)$  w.p.  $p_{t,t'}$  and put  $\sigma(i+1)$  ahead of  $\sigma(i)$  w.p.  $1 - p_{t,t'}$ . We denote the linear particle system having  $n$  particles of  $k$  different types by  $\mathcal{X}_k(n)$ .

This chain is also a reversible Markov chain. In a special case where  $p_{t,t'} = \frac{s(t)}{s(t)+s(t')}$  the stationary distribution  $\pi$  is

$$\pi(\sigma) = \prod_{i=1}^n s(i)^{\sigma^{-1}(i)} / Z'. \quad (Z' \text{ is a normalizing factor.}) \quad (2)$$

► **Proposition 3.** *By regarding gladiators of equal strength as indistinguishable particles, we associate to any gladiator system a linear particle system.*

Note that the state space of the gladiator system has cardinality  $n!$  for  $n$  different gladiators but the linear particle system has only  $n!/(n_1!n_2! \dots n_k!)$  states, since particles of the same type are indistinguishable. Thus,  $Z' \ll Z$ . The following theorem, whose proof will be presented later, shows the connection between the mixing times of the two chains.

► **Theorem 4.** *Let  $t(\mathcal{X}_k)$  and  $t(\mathcal{G}_k)$  be respectively the mixing times for a linear particle system and its corresponding gladiator chain. Then,  $t(\mathcal{G}_k) \leq \mathcal{O}(n^8) t(\mathcal{X}_k)$ .*

Our main result, which extends the results of Benjamini et al. [1] on exclusion processes, is the following:

► **Theorem 5.** *Let  $\mathcal{X}_3(n)$  be a linear particle system of Definition 2, having particles of type  $A$ ,  $B$  and  $C$ . Assume that we have strength functions assigned to each particle type, namely  $s_A < s_B < s_C$ , and thus swapping probabilities  $p_{B,A} = s_A/(s_A + s_B)$ ,  $p_{B,C} = s_C/(s_C + s_B)$  and  $p_{A,C} = s_C/(s_A + s_C)$ . If  $s_A/s_B, s_B/s_C < 1/2$ , then the mixing time of  $\mathcal{X}_3(n)$  satisfies  $t(\mathcal{X}_3(n)) \leq \mathcal{O}(n^{10})$ .*

► **Remark.** The condition  $s_A/s_B, s_B/s_C \leq 1/2$  comes from the following simple bound on  $q$ -binomials that the reader should be able to verify easily: If  $q < 1/2$  then,  $\binom{m}{r}_q < 2^r < (\frac{1}{q})^r$ . Better bounds on  $q$ -binomials would allow the result to be improved.

We will prove Theorem 5 in Section 3. Having Theorem 5, we deduce the following case of Fill's conjecture:

► **Corollary 6.** *The mixing time of  $\mathcal{G}_3(n)$  satisfies  $t(\mathcal{G}_3(n)) \leq \mathcal{O}(n^{18})$ , if  $s_A/s_B < 1/2$  and  $s_B/s_C < 1/2$ , where  $C$  is the strongest playing team among the three, and the gladiators in team  $B$  are stronger than the gladiators in team  $A$ .*

**Proof.** From Theorems 5 and 4. ◀

► **Corollary 7.** (*Generalization of league hierarchies*) Let  $T$  be a ternary tree with  $n$  leaves. The children of each interior node  $v$  are labeled with labels  $A(v)$ ,  $B(v)$ , and  $C(v)$ , and they have strength values  $s_{A(v)}$ ,  $s_{B(v)}$ , and  $s_{C(v)}$ . The leaves are labeled by numbers  $1, 2, \dots, n$ . The probability of putting  $j$  ahead of  $i$  for  $j > i$  is equal to  $p_{i,j} = s_{X(v)}/(s_{X(v)} + s_{Y(v)})$  where  $v$  is the node that is the lowest common ancestor of  $i$  and  $j$  in  $T$ , and  $X(v)$  is the child of  $v$  which is an ancestor of  $j$ , and  $Y(v)$  is the child of  $v$  which is an ancestor of  $i$ . If for each  $v \in T$ ,  $s_{A(v)}$ ,  $s_{B(v)}$ , and  $s_{C(v)}$  satisfy the conditions in Theorem 5, then the mixing time of the league hierarchy chain is  $\mathcal{O}(n^{10} \log n)$ .

**Proof.** A correspondence between the league hierarchies for binary trees (having  $n$  leaves) and a product of  $n-1$  copies of the simple exclusion processes is presented in [2]. Using this correspondence and employing Benjamini's result ([1]), Bhakta et al prove that the binary tree league hierarchies mixes rapidly. The correspondence introduced in [2] is in fact a correspondence between the  $k$ -ary league hierarchy chains and  $\mathcal{X}_k$ , and they prove that the mixing times of the league hierarchy and the linear particle system are related. Theorem 5 can be used to extend the results in [2] to ternary trees satisfying the restrictions. ◀

We finish this section by proving Theorem 4.

## 2.1 Gladiators and Particles (Proof of Theorem 4)

Consider the gladiator chain  $\mathcal{G}_k(n)$  for arbitrary  $n$  being the number of gladiators and  $k$  the number of teams. Assume that we have  $n_i$  gladiators playing at team  $i$ ; hence,  $\sum_{i=1}^k n_i = n$ . At each step of the chain, one of two things is happening:

1. Whisking: gladiators of the same team are fighting.
2. Sifting: gladiators of different teams are fighting.

If we were restricted to whisking steps the chain would be equivalent to a product of several simple chains analyzed by Wilson [15]. If we were restricted to sifting steps the chain would be the linear particle system chain introduced in Definition 2. In order to study the mixing time of the gladiator chain we analyze sifting and whisking steps separately, and then we employ the following decomposition theorem:

► **Theorem 8** (Decomposition Theorem [9]). Let  $\mathcal{M}$  be a Markov chain on state space  $\Omega$  partitioned into  $\Omega_1, \Omega_2, \dots, \Omega_k$ . For each  $i$ , let  $\mathcal{M}_i$  be the restriction of  $\mathcal{M}$  to  $\Omega_i$  that rejects moves going outside of  $\Omega$ . Let  $\pi_i(A) = \pi(A \cap \Omega_i)/\pi(\Omega_i)$  for  $A \subseteq \Omega_i$ . We define the Markov chain  $\bar{\mathcal{M}}$  on state space  $\{1, \dots, k\}$  as follows:  $Pr_{\bar{\mathcal{M}}}(i, j) = \sum_{x \in \Omega_i, y \in \Omega_j} \pi_i(x) Pr_{\mathcal{M}}(x, y)$ , where  $Pr_{\mathcal{M}}$  and  $Pr_{\bar{\mathcal{M}}}$  are transition probabilities of  $\mathcal{M}$  and  $\bar{\mathcal{M}}$  respectively. Then:

$$t(\mathcal{M}) \leq 2t(\bar{\mathcal{M}}) \max_i \{t(\mathcal{M}_i)\}.$$

To apply the decomposition theorem, we partition  $S_n$  to  $S_{\sigma_1, \sigma_2, \dots, \sigma_k}$  for all choices of  $\sigma_1 \in S_{n_1}, \sigma_2 \in S_{n_2}, \dots, \sigma_k \in S_{n_k}$ ; each  $S_{\sigma_1, \sigma_2, \dots, \sigma_k}$  being the set of all permutations in  $S_n$  in which all the gladiators corresponding to particle  $i$  preserve the ordering associated to them by  $\sigma_i$ . The restriction of  $\mathcal{G}_k(n)$  to  $S_{\sigma_1, \sigma_2, \dots, \sigma_k}$  is equivalent to  $\mathcal{X}_k(n)$ . We define  $\bar{\mathcal{G}}$  to be the Markov chain on  $\prod_{i=1}^k S_{n_i}$  with the following transition probabilities:

$$Pr_{\bar{\mathcal{G}}}(S_{\sigma_1, \sigma_2, \dots, \sigma_i, \dots, \sigma_k}, S_{\sigma_1, \sigma_2, \dots, \sigma'_i, \dots, \sigma_k}) = \frac{1}{\pi(S_{\sigma_1, \sigma_2, \dots, \sigma_i, \dots, \sigma_k})} \sum_{\substack{x \in S_{\sigma_1, \sigma_2, \dots, \sigma_i, \dots, \sigma_k}, \\ y \in S_{\sigma_1, \sigma_2, \dots, \sigma'_i, \dots, \sigma_k}}} \pi(x) Pr_{\mathcal{G}}(x, y),$$

## 41:6 Mixing of Permutations by Biased Transposition

where  $\sigma_i$  and  $\sigma'_i$  are only different in swapping  $j$  and  $j+1$ st elements and  $Pr_{\mathcal{G}}(x, y) = 1/2(n-1)$  iff  $j$  and  $j+1$ st copies of particle  $i$  are adjacent in  $x$  and swapped in  $y$ . Moreover, we observe that:

$$\frac{1}{\pi(S_{\sigma_1, \sigma_2, \dots, \sigma_i, \dots, \sigma_k})} \sum_{\substack{x \in S_{\sigma_1, \sigma_2, \dots, \sigma_i, \dots, \sigma_k}, \\ y \in S_{\sigma_1, \sigma_2, \dots, \sigma'_i, \dots, \sigma_k}}} \pi(x) \geq 1/(n-1).$$

We can verify the above equation by the following reasoning: consider an arbitrary permutation  $z \in S_{\sigma_1, \sigma_2, \dots, \sigma_i, \dots, \sigma_k}$  in which  $j$ th and  $j+1$ st copies of particle  $i$  are not adjacent. We can map  $z$  to two other permutations  $z_1$  and  $z_2$  where in  $z_1$  we take the the  $j$ th copy of particle  $i$  down to make it adjacent to the  $j+1$ st copy, and in  $z_2$  we take the the  $j+1$ st copy of particle  $i$  up to make it adjacent to the  $j$ th copy. We will have  $\pi(z)/\pi(z_1) = \pi(z_2)/\pi(z)$ , and hence one of  $\pi(z_1)$  or  $\pi(z_2)$  will be larger than  $\pi(z)$ . This mapping is in worst case  $n-1$  to 1, hence the above equation holds.

Having the above observations, we realize  $\bar{\mathcal{G}}$  is the product of  $k$  adjacent transposition Markov chains, and in each of these Markov chains we swap two adjacent elements with probability at least  $1/2(n-1)^2$ . Let these chains be  $\bar{\mathcal{G}}_1, \bar{\mathcal{G}}_2, \dots, \bar{\mathcal{G}}_k$ . By comparing the conductance (for more information about conductance, see [8]) of this chain to the simple chain analyzed by Wilson [15], for each  $i$  we will have  $t(\bar{\mathcal{G}}_i) \leq n_i^8$ . We know by a result of Bhakta et al. [2] that if  $\bar{\mathcal{G}}$  is a product of  $k$  independent Markov chains  $\{\bar{\mathcal{G}}_i\}_{i=1}^k$  and it updates each  $\bar{\mathcal{G}}_i$  with probability  $p_i$ , then

$$t_\epsilon(\bar{\mathcal{G}}) \leq \max_{i=1, \dots, n} \frac{2}{p_i} t_{\frac{\epsilon}{2k}}(\bar{\mathcal{G}}_i).$$

Plugging in  $p_i = n_i/n$ , we have  $t(\bar{\mathcal{G}}) \leq \max(2n/n_i)n_i^8 \leq 2n^8$ . Summing up and employing the Decomposition Theorem,

$$t(\mathcal{G}_k(n)) \leq 4n^8 t(\mathcal{X}_k(n)).$$

### 3 Three Particle Systems (Proof of the Main Theorem)

In this section we prove Theorem 5 which states that  $t(\mathcal{X}_3(n)) \leq \mathcal{O}(n^{10})$  if  $s_A/s_B, s_B/s_C \leq 1/2$ .

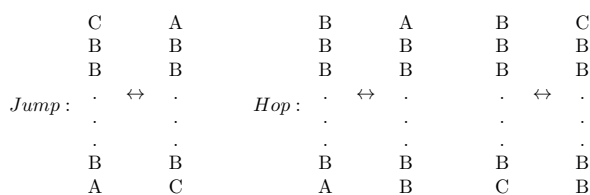
Assume that we have  $a$  copies of particle  $A$ ,  $b$  copies of particle  $B$ , and  $c$  copies of particle  $C$ . We denote the set containing all the different arrangements of these particles by  $\Omega_{a,b,c}$ . We introduce another Markov chain  $\mathcal{X}_t(n)$  on the same sample space  $\Omega_{a,b,c}$ . Using the comparison method (see [12]) we will show that the mixing times of  $\mathcal{X}_3(n)$  and  $\mathcal{X}_t(n)$  are related.

Then we will use the path congestion technique to show  $\mathcal{X}_t(n)$  mixes in polynomial time, and hence we deduce Theorem 5.

$$\text{mixing time of } \mathcal{X}_3(n) \xleftarrow[\text{technique}]{\text{Comparison}} \text{mixing time of } \mathcal{X}_t(n)$$

**Notation.** We denote the substring  $\sigma(i)\sigma(i+1)\dots\sigma(j)$  by  $\sigma[i, j]$ , and by  $B^t$  we refer to a string which is  $t$  copies of particle  $B$ .

► **Definition 9.** Let  $\mathcal{X}_t(n)$  be a Markov chain on state space  $\Omega_{a,b,c}$  and  $n = a + b + c$ . If the current state is  $\sigma$ , we choose natural numbers  $1 \leq i < j \leq n$  uniformly at random and swap them following these rules (Figure 1):



■ **Figure 1** Jumps and Hops are the transitions in the Markov chain  $\mathcal{X}_t$ .

1. If  $\sigma(i) = A$  and in  $\sigma(j) = C$  or vice versa and  $\sigma[i+1, j-1] = B^{j-i-1}$ . Then, put  $\sigma(i)$  and  $\sigma(j)$  in increasing order of their strength w.p.  $(s_C/s_A)^{j-i}/(1 + (s_C/s_A)^{j-i})$ . With probability  $1/(1 + (s_C/s_A)^{j-i})$ , put them in decreasing order. We call this move a *Jump* and we denote it by  $\mathcal{J}_i^j(A, C)$  if  $\sigma(i) = A$  and  $\sigma(j) = C$ ; and  $\mathcal{J}_i^j(C, A)$  for vice versa.
2. If  $\sigma[i, j-1] = B^{j-i}$  and  $\sigma(j) = A$  or if  $\sigma[i+1, j] = B^{j-i}$  and  $\sigma(i) = A$ . Then, put  $\sigma(i)$  and  $\sigma(j)$  in increasing order of their strength w.p.  $(s_B/s_A)^{j-i}/(1 + (s_B/s_A)^{j-i})$ . With probability  $1/(1 + (s_B/s_A)^{j-i})$ , put them in decreasing order. We call this move a *Hop*, and we denote it by  $\mathcal{H}_i^j(A, B)$  if  $\sigma(i) = A$  and  $\sigma(j) = B$ ; and  $\mathcal{H}_i^j(B, A)$  for vice versa. Similar rules and notation apply when swapping  $B$  and  $C$ .
3. Else, do nothing.

It can be easily checked that  $\mathcal{X}_t$  is reversible and its stationary distribution is the  $\pi$  in Equation 2.

► **Lemma 10.**  $t(\mathcal{X}_3(n)) \leq 2n^4 t(\mathcal{X}_t(n))$ .

**Proof.** We use the comparison technique<sup>4</sup> in the proof of Lemma 10 (see [3, 12]). To any edge  $(\sigma, \tau)$  in  $\mathcal{X}_t$ , we correspond a path from  $\sigma$  to  $\tau$  in  $\mathcal{X}_3$ . Let  $e_i(p, p')$  be a move in  $\mathcal{X}_3$  which swaps particles  $p$  and  $p'$  located at positions  $i$  and  $i + 1$  in an arrangement. To  $e = (\sigma, \tau)$  making  $\mathcal{J}_i^j(A, C)$  in  $\mathcal{X}_t$ , we correspond the following path in  $\mathcal{X}_3$ :  $e_i(A, B), e_{i+1}(A, B), \dots, e_{j-2}(A, B), e_{j-1}(A, C), e_{j-2}(B, C), \dots, e_i(B, C)$ . We denote this path by  $\gamma_{\sigma, \tau}$ , and the set containing all such paths by  $\Gamma_{\mathcal{J}}$ . Similarly, to  $e = (\sigma, \tau)$  making  $\mathcal{H}_i^j(A, B)$  in  $\mathcal{X}_t$ , we correspond the following path in  $\mathcal{X}_3$ :  $e_i(A, B), e_{i+1}(A, B), \dots, e_{j-2}(A, B), e_{j-1}(A, B)$ . We denote this path by  $\gamma_{\sigma, \tau}$ , and the set containing all such paths by  $\Gamma_{\mathcal{H}}$ . Let  $\Gamma = \{\gamma_{\sigma, \tau}\}_{\sigma, \tau \in \Omega_{a,b,c}} = \Gamma_{\mathcal{J}} \cup \Gamma_{\mathcal{H}}$ .

We now bound the congestion placed by  $\Gamma$  on edges of  $\mathcal{X}_3$ . Consider an arbitrary  $e = (\alpha, \beta)$  making swap  $e_i(A, B)$  and assume  $\alpha[i - t - 1, i + d + 1] = pB^tAB^dp'$  where  $p$  and  $p'$  are particles different from  $B$ . For any  $\sigma$  and  $\tau$  in  $\Omega_{a,b,c}$  if  $e \in \gamma_{\sigma, \tau}$  then, there must be  $i - t \leq j \leq i - 1$  and  $i + 1 \leq k \leq i + d$  such that  $\gamma_{\sigma, \tau}$  corresponds to  $\mathcal{H}_j^k(A, B)$  or it corresponds to  $\mathcal{J}_j^{i+d+1}(A, p')$ . Thus, the congestion placed on  $e$  only by paths in  $\Gamma_{\mathcal{H}}$  is:

$$\begin{aligned} \frac{\sum_{\{\sigma, \tau | e \in \gamma_{\sigma, \tau} \in \Gamma_{\mathcal{H}}\}} |\gamma_{\sigma, \tau}| \mathcal{C}(\sigma, \tau)}{\mathcal{C}(e)} &= \sum_{j=i-t}^{i-1} \sum_{k=i+1}^{i+d} \frac{|\gamma_{\sigma, \tau}| (s_B/s_A)^{i-j} (1 + s_B/s_A)}{(1 + (s_B/s_A)^{k+1-j})} \\ &\leq 2(d+t) \sum_{j'=1}^t \sum_{k'=1}^d \frac{(s_B/s_A)^{j'} (s_B/s_A)}{(1 + (s_B/s_A)^{j'+k'})} \leq 2t(d+t) \sum_{k'=1}^d \frac{s_B/s_A}{(s_B/s_A)^{k'}} \leq n^2. \end{aligned}$$

<sup>4</sup> The comparison method was introduced by Diaconis and Saloff-Coste [3] and then Randall and Tetali extended it and employed it for analysis of Glauber dynamics [12]. For more information about this method we encourage the reader to refer to [8].

## 41:8 Mixing of Permutations by Biased Transposition

We can similarly show that the congestion placed on  $e$  by  $\Gamma_{\mathcal{J}}$  is less than  $n^2$ , where  $n$  is the length of the arrangements or total number of particles. For each  $e \in E(\mathcal{M})$ , let  $\mathcal{A}_e$  be the congestion  $\Gamma$  places on  $e$ . We will have  $\mathcal{A}_e \leq 2n^2$ . We also know  $\pi_{\min} \geq q^{n(n+1)}$ , where  $q = \max\{s_A/s_B, s_B/s_C\}$  (we consider  $q$  to be a constant). Hence, employing the Comparison Theorem we have

$$t(\mathcal{X}_3(n)) \leq \max_{e \in E(\mathcal{M})} \mathcal{A}_e \pi_{\min} t(\mathcal{X}_t(n)) \leq (2n^4)t(\mathcal{X}_t(n)). \quad \blacktriangleleft$$

Having the above connection it suffices to bound the mixing time of  $\mathcal{X}_t(n)$ . In the rest of this section our goal is to bound  $t(\mathcal{X}_t)$ . We use the path congestion theorem, which is stated below. In particular, for any two arbitrary states  $\sigma, \tau \in \Omega_{a,b,c}$ , we introduce a path  $\gamma_{\sigma,\tau}$ . Then we show that none of the edges of the Markov chain  $\mathcal{X}_t(n)$  is congested heavily by these paths. Formally, we employ Theorem 11 and in Theorem 13 we show that  $t(\mathcal{X}_t(n)) \leq \mathcal{O}(n^4)$ .

► **Theorem 11** (Canonical Paths Theorem [7]). *Let  $\mathcal{M}$  be a Markov chain with stationary distribution  $\pi$  and  $E$  the set of the edges in its underlying graph. For any two states  $\sigma$  and  $\tau$  in the state space  $\Omega$  we define a path  $\gamma_{\sigma,\tau}$ . The congestion factor for any edge  $e \in E$  is denoted by  $\Phi_e$  and is defined by  $\Phi_e = \frac{1}{C(e)} \sum_{x,y \in \gamma_{\sigma,\tau}} \pi(x)\pi(y)$ . We can bound the mixing time of  $\mathcal{M}$  using the congestion factor:  $t_\epsilon(\mathcal{M}) \leq 8\Phi^2(\log \pi_{\min}^{-1} + \log \epsilon)$ , where  $\Phi = \max_{e \in E} \Phi_e$ ,  $\pi_{\min} = \min_{x \in \Omega} \pi(x)$  and  $\epsilon$  is the convergence parameter.*

**The Paths.** For each  $\sigma, \tau \in \Omega_{a,b,c}$ , we introduce the following path in  $\mathcal{X}_t$  from  $\sigma$  to  $\tau$ : We partition  $\sigma$  and  $\tau$  to  $b+1$  blocks; the end points of these blocks are locations of  $B$ s in  $\tau$ . For instance if in  $\tau$ , the first  $B$  is located at position  $i$  and the second  $B$  is located at position  $j$  then, the first block in both  $\sigma$  and  $\tau$  is  $[1, i]$ , and the second is  $[i+1, j]$ . Starting from the first block, we change each block in two steps, first we use Jump moves and change the relative position of  $A$  and  $C$ s in  $\sigma$  to become in the order in which they appear in  $\tau$ . Then, we bring the  $B$  in that block to its location in  $\tau$ . Formally, we repeat the following loop:

**Notation.** By saying  $k = B_j(\sigma)$ , we mean the  $j$ th copy of particle  $B$  is located at position  $k$  in  $\sigma$ .

Starting from  $\sigma$ , we repeat the following steps until  $\tau$  is reached.

Initially, let  $i, j = 1$ .

1. Let  $k = B_j(\tau)$ . We define the  $j$ th block of  $\sigma$  and  $\tau$  to be the substring starting from  $i$  and ending in  $k$ . Note that in  $\tau$ , each blocks starts right after a  $B$  and ends with a  $B$ . In the  $j$ th iteration, the goal is to change  $\sigma[i, k]$  until  $\sigma[1, k] = \tau[1, k]$ , i.e. the first  $j$  blocks equal in  $\sigma$  and  $\tau$ .
2. Using Jumps, and starting from the lowest index  $i$ , we bring particles  $C$  or  $A$  down until  $A$  and  $C$  particles in the block  $[i, k]$  have the same order in  $\sigma$  and  $\tau$ .
3. We use Hops and bring the  $j$ th  $B$  in  $\sigma$  to  $B_j(\tau)$ . In this process, we may need to bring several copies of particle  $B$  out of the  $j$ th block in  $\sigma$ . In that case, we choose a random ordering of  $B$ s and move them with respect to that order (details explained in the proof of Claim 12).
4. Set  $i = B_j(\tau) + 1$ .
5. Increment  $j$ .

► **Claim 12.** *Let  $\{\gamma_{\sigma,\tau}\}_{\sigma,\tau \in \Omega_{a,b,c}}$  be the set of paths defined as above. Then, for any arbitrary edge  $e$  in the Markov chain  $\mathcal{X}_t$  the congestion  $\Phi_e$ , defined in Theorem 11 satisfies  $\Phi_e \leq n$ .*



	1st iteration:				2nd iteration:				3rd iteration:														
$\sigma :$	C	C	C	C	C	C	C	C	C	C	C	C	$\tau :$	A	C	C	B	B	B	A	A	C	C
	A	A	A	A	A	A	A	A	A	A	A	A		C	C	C	C	C	C	C	C	C	
	B	B	B	B	B	B	B	B	B	B	B	B		C	C	C	C	C	C	C	C	C	
	C	C	C	C	C	C	C	C	C	C	C	C		B	B	B	B	B	B	B	B	B	
	A	Jump	A	Jump	A	Jump	A	Hop	A	Hop	B	Hop	B	Jump	B								
	C	→	C	→	A	→	A	→	B	→	A	→	A	→	A								
	B	Step 1	B	Step 1	B	Step 1	B	Step 2	A	Step 2	A	Step 2	A	Step 1	A								
	C		A		C		C		C		C		C		C								
	A		C		C		C		C		C		C		C								

Figure 2 We use the path congestion technique to bound  $t(\mathcal{X}_t)$ . In each iteration we fix a block in  $\sigma$  until  $\tau$  is reached.

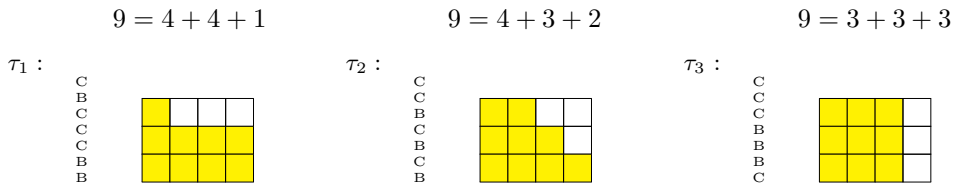


Figure 3 Correspondence of partition functions with q-binomials: There are three integer partitions of 9 that fit into a  $3 \times 4$  rectangle, and there are three arrangements of gladiators in  $\Omega_{0,3,4}$  with  $q(\tau_1) = q(\tau_2) = q(\tau_3) = q^9$ . i.e. the coefficient for  $q^9$  in  $\binom{7}{3}_q$  equals 3.

**Proof.** In order to verify the claim, we analyse the congestion of Jump and Hop edges separately. In both of the analyses, we consider an edge  $e = (\alpha, \beta)$ , and to any  $\sigma, \tau$  such that  $e \in \gamma_{\sigma, \tau}$  we assign a  $\mathcal{F}_e(\sigma, \tau) \in \Omega_{a,b,c}$ . The reverse image of  $\mathcal{F}$  could be a subset of  $\Omega_{a,b,c} \times \Omega_{a,b,c}$ . However, using  $q$ -binomials<sup>5</sup> we show that  $\sum_{\sigma, \tau}$  are mapped to the same  $\zeta \pi(\sigma)\pi(\tau)$  is bounded by a polynomial function of  $n$  multiplied by  $\pi(\zeta)$ , and then we conclude the claim. A key factor of our analysis is the use of  $q$ -binomials. Note the following observations: Assume that we have no copies of particle  $A$ ,  $b$  copies of  $B$ , and  $c$  copies of particle  $C$ . Let  $M \in \Omega_{0,b,c}$  be the arrangement with maximum stationary probability, i.e.  $M = B^b C^c$ . Note that for each  $\sigma \in \Omega_{0,b,c}$ ,  $\pi(\sigma)/\pi(M) = (s_B/s_C)^t$ , where  $t$  is the number of transpositions needed to get from  $M$  to  $\sigma$ . For a constant  $t$ , the number of  $\sigma$ s requiring  $t$  transpositions is equal to the number of integer partitions of  $t$  fitting in an  $b \times c$  rectangle (see Figure 3). Thus:

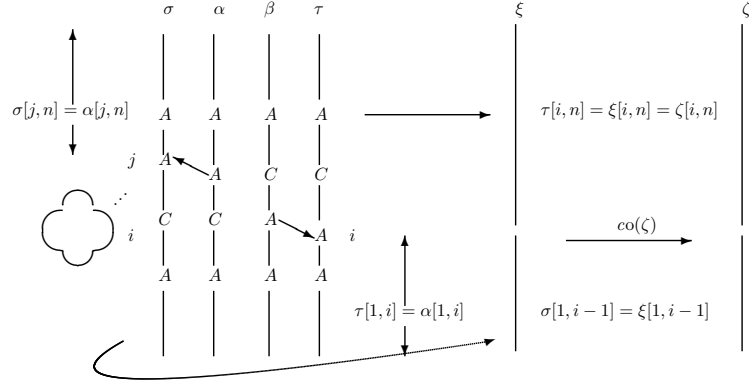
$$\sum_{\sigma \in \Omega_{0,b,c}} \frac{\pi(\sigma)}{\pi(M)} = \binom{b+c}{b}_q ; q = s_B/s_C.$$

Consider an edge  $e = (\alpha, \beta)$  corresponding to  $\mathcal{J}_k^{k+g}(C, A)$ . Assume that  $k = C_l(\alpha)$ ,  $k + d = A_m(\alpha)$  (remember the notation  $k = p_m(\sigma)$  meaning the  $m$ th copy of particle  $p$  is located at position  $k$  in  $\sigma$ ), i.e. this edge is swapping the  $l$ th  $C$  with the  $m$ th  $A$  in  $\alpha$ .

It follows from the way we set the paths that, for some  $j$ ,  $A_m(\alpha) \leq j < A_{m+1}(\alpha)$ ,  $A_m(\sigma) =$

<sup>5</sup> More information about  $q$ -binomials can be found in Richard Stanley’s course “Topics in Algebraic Combinatorics,” Chapter 6 (see [14]). Here we use the the following bound which can be simply verified by expanding the formula for  $q$ -binomials: if  $q < 1/2$  then,  $\binom{m}{r}_q < \prod_{i=1}^r 1/(1-q) < 2^r < (\frac{1}{q})^r$ .

41:10 Mixing of Permutations by Biased Transposition



■ **Figure 4** We define  $\mathcal{F}_e(\sigma, \tau) = \zeta$ . To produce  $\zeta$  we first concat  $\sigma[1, i-1]$  and  $\tau[i, n]$  then, we substitute some particles.

$j$  and for some  $i, A_{m-1}(\beta) < i \leq A_m(\beta)$ ,  $A_m(\tau) = i$ . The preceding blocks of  $\alpha$  have been changed in accordance with  $\tau$ , and the succeeding blocks of  $\alpha$  have not been changed yet, hence they resemble  $\sigma$  blocks. Therefore we have  $\alpha[1, i-1] = \tau[1, i-1]$  and  $\alpha[j+1, n] = \sigma[j+1, n]$  (see Figure 4).

We define the function  $\mathcal{F}_e : \Omega_{a,b,c} \times \Omega_{a,b,c} \rightarrow \Omega_{a,b,c}$  as follows: For any  $\sigma, \tau$  satisfying  $e \in \gamma_{\sigma, \tau}$ , let  $\xi_{\sigma, \tau} := \sigma[1, i-1]|\tau[i, n]$  (the symbol  $|$  denotes concatenation.) Since the arrangements of particles is changing we may have  $\xi_{\sigma, \tau} \notin \Omega_{a,b,c}$ . For instance we may have  $\tau[i, n] \in \Omega_{x,y,z}$  and  $\sigma[1, i-1] \in \Omega_{x',y',z'}$  but  $x+x' \neq a$  or  $y+y' \neq b$  or  $z+z' \neq c$ . However, we know  $a - (x+x') + (b - (y+y')) + (c - (z+z')) = 0$ , which means there is a way to substitute the particles in  $\sigma[1, i-1]$  to change  $\xi$  to  $\zeta$  so that  $\zeta \in \Omega_{a,b,c}$ . We call this stage the substitution stage, in which we identify the particle or particles with extra copies in  $\sigma[1, i-1]$ , and we substitute the lowest copies of them with inadequate particles and produce  $\zeta \in \Omega_{a,b,c}$ . Then, we define  $\mathcal{F}_e(\sigma, \tau) := \zeta$ . For instance, if  $a - (x+x') + (b - (y+y')) = -(c - (z+z'))$ , then substitute the lowest  $c - (z+z')$  copies of  $A$  and  $B$  with  $C$ s, and produce  $\mathcal{F}_e(\sigma, \tau) = \zeta$ . The substitution stage will cause a *substitution cost*, we denote the substitution cost by  $co(\zeta)$ , and define it as:  $co(\zeta) = \pi(\zeta)/\pi(\xi)$ , where  $\xi = \sigma[1, i-1]|\tau[i, n]$ . Note that if we make  $t$  substitutions, the substitution cost is at most  $(s_C/s_A)^t$ . To make the analysis simpler we only analyze the worst case in which we assume we have substituted  $t$   $C$ s with  $A$ s in  $\sigma[1, i-1]$ . This assumption also means that in  $\sigma[i, j]$  we have  $t$  more  $A$ s and  $t$  fewer  $C$ s than in  $\alpha[i, j]$ .

Consider  $\sigma, \tau$  such that  $e \in \gamma_{\sigma, \tau}$ . Let  $\mathcal{F}_e(\sigma, \tau) = \zeta$ . We have,

$$\frac{\pi(\zeta)}{\pi(\alpha)} = \left( \frac{\pi(\tau)}{\pi(\alpha)} \right) \left( \frac{\pi(\sigma)}{\pi(\alpha)} \right) \left( \frac{w^i(\alpha[i, j])}{w^i(\sigma[i, j])} \right) co(\zeta),$$

where the later term is the substitution cost, and  $w^i(\sigma[i, j]) := \prod_{k=i}^j s(k)^{i+\sigma^{-1}(k)}$ . Having  $g = A_m(\alpha) - C_l(\alpha)$  we will get:

$$\Phi_e = (1 + (s_A/s_C)^g) \left( \sum_{\sigma; \alpha[j+1, n] = \sigma[j+1, n]} \frac{\pi(\sigma)}{\pi(\alpha)} \sum_{\tau; \alpha[1, i-1] = \tau[1, i-1]} \frac{\pi(\tau)}{\pi(\alpha)} \right) \pi(\alpha)$$

Let  $\mathcal{S}_t$  be the set of all  $\sigma$ s with  $t$  substitutions. We have:

$$\Phi_e \leq \sum_{\substack{\zeta \text{ needs } t \\ \text{substitutions}}} \frac{1}{co(\zeta)} \sum_{\tau} \sum_{\sigma \in \mathcal{S}_t} \left( \frac{\pi(\mathcal{F}_e(\sigma, \tau))}{\pi(\alpha)} \right) \left( \frac{w^i(\sigma[i, j])}{w^i(\alpha[i, j])} \right) \pi(\alpha).$$

Let  $M_t(\alpha)$  be the arrangement that we get from replacing the lowest  $t$  copies of particle  $C$  with copies of particle  $A$  in  $\alpha[i, j]$ . We have:  $\sum_{\sigma \in S_t} \frac{w^i(\sigma[i, j])}{w^i(\alpha[i, j])} = \frac{w^i(M_t)Q_{\bar{B}}^i(M_t(\alpha))}{w^i(\alpha[i, j])}$ , where  $w^i(\sigma[i, j]) := \prod_{k=i}^j s(k)^{i+\sigma^{-1}(k)}$ , and  $Q_{\bar{B}}^i(M_t(\alpha)) := \sum_{\sigma: \text{fix the positions of all Bs in } M_t(\alpha) \text{ and rearrange the rest of particles}} \pi(\sigma)/\pi(M_t(\alpha))$ .

Note that  $w^i(M_t)Q_{\bar{B}}(M_t) \leq q^{t(t+1)-2t}w^i(\alpha[i, j])$ , where  $q = \max\{s_A/s_B, s_B/s_C\}$ . This inequality holds because  $Q_{\bar{B}}(M_t) \leq \binom{y}{t}_{s_A/s_C} \leq q^{-2t}$  and  $w(M_t)/w(\alpha[i, j]) \leq q^{t(t+1)}$ .

Moreover,  $\sum_{\text{substitutions}} \zeta \text{ needs } t \frac{1}{\text{co}(\zeta)} \leq \binom{t+b'}{t} q^2 \leq q^{-2t}$ , where  $b'$  is the number of  $B$ s in  $\sigma[0, i-1]$  and  $q = \max\{s_A/s_B, s_B/s_C\}$ .

Putting all of the above inequalities together, we will have that each edge of Move 2 is only congested by:

$$\Phi_e \leq (1 + q^g) \sum_t (q^{t(t+1)-4t}) \leq n.$$

So far, we showed that any Jump edge is only congested by a factor of a polynomial function of  $n$ . Consider an edge corresponding to a Hop, namely  $e$ . We denote this edge by  $e = (\alpha, \beta)$ . Assume we are swapping  $A$  and  $B$ .

Consider a state  $\sigma$  traversing  $e$  to get to  $\tau$ , and assume we traversed  $e$  while fixing block  $[i, j]$ . Since we are making a Hop,  $A$ s and  $C$ s in the block are fixed according to  $\tau$ , and we are bringing the  $k$ th  $B$  to its position in  $\tau$ .

Before we proceed to the proof there is a subtlety about using a Hop that needs to be explained. If  $A_k$  has to go down to reach its position in  $\tau$  or if there is only one copy of it in the block there is no complication. Let's assume we have  $t$  copies of particle  $B$  in  $\sigma[i, j]$ . All of the  $t$  copies of  $B$  should move up and stand out of block  $\sigma[i, j]$  to reach their position in  $\tau$ . In order to accomplish this, we choose a subset  $S$  of  $\{1_k, \dots, 1_{t+k}\}$  uniformly at random and we move the elements of  $S$  in decreasing order of their index out of the block.

Assume, when going from  $\sigma$  to  $\tau$  we used  $e = (\alpha, \beta)$  and in  $\alpha[i, j]$  we have  $t$  copies of particle  $B$ :  $B_k, \dots, B_{k+t}$  and swapping  $B_{k+l}, B_{k+l+1}, \dots, B_{k+d}$  with the next  $A$ . We have,  $\tau[1, i] = \alpha[1, i]$ ,  $\sigma[j+t, n] = \alpha[j+t, n]$ , and for any  $i$  if  $B_{k+i}(\alpha) < B_{l+k}(\alpha)$  then,  $B_{k+i}(\alpha) = B_{k+i}(\sigma)$ . The following information about  $S$  can be determined by examining  $\alpha$  and  $\beta$ :  $B_{k+d+1}, \dots, B_{k+t} \notin S$  while  $S$  may contain any of  $B_k, \dots, B_{k+l}$ . Therefore, among the random paths connecting  $\sigma$  to  $\tau$ , there are  $2^l$  subsets traversing through  $e$  and hence the congestion they place on  $e$  is  $\pi(\tau)\pi(\sigma)/2^{t-l}$ .

To bound  $\Phi_e$  for each  $e$  we introduce correspondence  $\mathcal{F}_e : \Omega_{a,b,c} \times \Omega_{a,b,c} \rightarrow \Omega_{a,b,c}$  satisfying:

$$\forall \zeta \in \mathcal{F}_e(\Omega_{a,b,c}); \frac{\sum_{\mathcal{F}_e^{-1}(\zeta)=(\sigma,\tau)} \pi(\sigma)\pi(\tau)}{\pi(\alpha)} \leq 2^{t-l}\pi(\zeta); \quad (3)$$

where  $c$  is the number of  $C$ s in  $\alpha[i, j]$  and  $\mathcal{F}_e(\sigma, \tau) \neq \text{NULL}$  if and only if,  $e = (\alpha, \beta) \in \gamma_{\sigma, \tau}$ .

Let  $\sigma$  and  $\tau$  be two ends of a path traversing through  $e$ , we define  $\mathcal{F}_e := \sigma[1, i-1]|\tau[i, n]$ , to verify Equation 3 take  $\zeta = \mathcal{F}_e(\sigma, \tau)$ . We have,  $\frac{\pi(\sigma)\pi(\tau)}{\pi(\alpha)} = \frac{\pi(\sigma)}{\pi(\alpha)} \frac{\pi(\tau)}{\pi(\alpha)} \pi(\alpha)$ . Thus,

$$\begin{aligned} \frac{\pi(\zeta)}{\pi(\alpha)} &= \frac{\pi(\zeta[1, i-1])}{\pi(\alpha[1, i-1])} \frac{\pi(\zeta[i, j-1])}{\pi(\alpha[i, j-1])} \frac{\pi(\zeta[j, n])}{\pi(\alpha[j, n])} = \frac{\pi(\sigma[1, i-1])}{\pi(\alpha[1, i-1])} \frac{\pi(\tau[i, j-1])}{\pi(\alpha[i, j-1])} \frac{\pi(\tau[j, n])}{\pi(\alpha[j, n])} \\ &= \frac{\pi(\sigma')}{\pi(\sigma)} \frac{\pi(\sigma)}{\pi(\alpha)} \frac{\pi(\tau)}{\pi(\alpha)}, \end{aligned}$$

where  $\sigma'$  is the following arrangement:  $\sigma' := \alpha[1, i-1]|\sigma[i, j-1]|\alpha[j, n]$ . We have  $\pi(\sigma')/\pi(\alpha) = \pi(\sigma[i, j])/\pi(\alpha[i, j])$ . Hence,

$$\sum_{\sigma, \tau; \mathcal{F}(\sigma, \tau) = \zeta} \frac{\pi(\sigma)\pi(\tau)}{\pi(\alpha)} = \sum_{\substack{\sigma, \tau \\ \mathcal{F}(\sigma, \tau) = \zeta}} \frac{\pi(\sigma')}{\pi(\sigma)} \pi(\zeta).$$

## 41:12 Mixing of Permutations by Biased Transposition

Since we have  $t - l$   $B$ s with undecided position between  $j - i$  other elements we have  $\sum \frac{\pi(\sigma')}{\pi(\sigma)} \leq \binom{j-i+t-l}{t-l}_q$ , where  $q = \max\{s_A/s_B, s_B/s_C\}$ . Thus, we have  $\sum \frac{\pi(\sigma')}{\pi(\sigma)} \leq 2^{t-l}$ . Hence, the congestion placed on  $e$  is:

$$\Phi_{e=(\alpha,\beta)} = (1 + q^g) \sum_{\substack{\sigma,\tau \\ e \in \gamma_{\sigma,\tau}}} \frac{\pi(\sigma)\pi(\tau)}{\pi(\alpha)2^{t-l}} \leq 1.$$

Summing up, we showed the for any arbitrary edge  $e$ ,  $\Phi_e \leq \max\{n, 1\}$ . ◀

Having the above claim, we now use the path congestion Theorem (Theorem 11) to bound  $t(\mathcal{X}_t(n))$ :

► **Theorem 13.** *If  $s_A/s_B, s_B/s_C \leq 1/2$ , then  $t(\mathcal{X}_t(n)) \leq \mathcal{O}(n^4)$ .*

**Proof.** Since  $\pi_{min} \geq q^{n(n+1)}$ ,  $q$  being maximum of  $s_A/s_B$  and  $s_B/s_C$ , we can apply Theorem 11 and we will have,  $t_\epsilon(\mathcal{X}_t) \leq 8n^2(n^2 + \ln(\epsilon^{-1})) \implies t(\mathcal{X}_t) \leq 8n^4$ . ◀

Finally, from Lemma 10 and Theorem 13 we conclude Theorem 5.

**Acknowledgements.** We would like to thank Dana Randall for a very helpful conversation about the gladiator problem and Fill's conjecture, and Sergi Elizalde for his help and knowledge concerning generating functions.

---

### References

- 1 I. Benjamini, N. Berger, C. Hoffman, and Mossel E. Mixing times of the biased card shuffling and the asymmetric exclusion process. *Transactions of the American Mathematical Society*, 357:3013–3029, 2005.
- 2 P. Bhakta, S. Miracle, D. Randall, and A. Streib. Mixing times of Markov chains for self-organized lists and biased permutations. *25th Symposium on Discrete Algorithms (SODA)*, 2014.
- 3 P. Diaconis and L. Saloff-Coste. Comparison techniques for random walks on finite groups. *The annals of applied probability*, 21:1149–1178, 1993.
- 4 P. Diaconis and M. Shahshahani. Generating a random permutation with random transpositions. *Probability theory and related fields*, 57:159–179, 1981.
- 5 J. Fill. An interesting spectral gap problem, 2003. Unpublished manuscript.
- 6 J. H. Hester and D. S. Hirscheberg. Self-organizing linear search. *ACM Computing survey*, 19:295–311, 1985.
- 7 M. Jerrum and Sinclair A. Approximating the permanent. *SIAM Journal on computing*, 18:1149–1178, 1989.
- 8 D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov chains and Mixing times*. American Mathematical Society, 2009.
- 9 R. Martin and D. Randall. Disjoint decomposition of Markov chains and sampling circuits in cayley graphs. *Combinatorics, Probability and Computing*, 15:411–448, 2006.
- 10 B. Morris. The mixing time for simple exclusion. *Annals of probability*, 16:63–67, 1976.
- 11 R. I. Oliveria. Mixing of the symmetric exclusion processes in terms of the corresponding single-particle random walk. *Annals of probability*, 41:871–913, 2013.
- 12 D. Randall and P. Tatali. Analyzing glauber dynamics by comparison of Markov chains. *Journal of Mathematical Physics*, 41:1598–1615, 2000.
- 13 R. Rivest. On self-organizing sequential search heuristics. *Communication of the ACM*, 19:63–67, 1976.

- 14 R. P. Stanley. Topics in algebraic combinatorics, 2012. Course notes for Mathematics, 2012. URL: <http://www-math.mit.edu/~rstan/algcomb/algcomb.pdf>.
- 15 D. Wilson. Mixing times of lozenge tiling and card shuffling Markov chains. *The annals of applied probability*, 1:274–325, 2004.



# Efficient Quantum Walk on the Grid with Multiple Marked Elements\*

Peter Høyer<sup>1</sup> and Mojtaba Komeili<sup>2</sup>

1 Department of Computer Science, University of Calgary, Calgary, Canada  
hoyer@ucalgary.ca

2 Department of Computer Science, University of Calgary, Calgary, Canada  
mojtaba.komeili@ucalgary.ca

---

## Abstract

We give a quantum algorithm for finding a marked element on the grid when there are multiple marked elements. Our algorithm uses quadratically fewer steps than a random walk on the grid, ignoring logarithmic factors. This is the first known quantum walk that finds a marked element in a number of steps less than the square-root of the extended hitting time. We also give a new tighter upper bound on the extended hitting time of a marked subset, expressed in terms of the hitting times of its members.

**1998 ACM Subject Classification** F.1.2 Modes of Computation, F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

**Keywords and phrases** Quantum walks, random walks, query complexity, spatial search

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.42

## 1 Introduction

Searching structured and unstructured data is one of the most fundamental tasks in computer science. In many search problems in quantum computing, we are given a set of  $N$  elements of which  $M$  elements are marked, and our task is to find and output a marked element.

Search problems have been studied intensively and found many applications, both classically and quantumly. The first result on search within quantum computing was given by Bennett et al. [9], who showed in 1994 that any quantum algorithm requires  $\Omega(\sqrt{N/M})$  steps to find a marked element. Grover [18] showed next that a quantum computer can find such a marked element in  $O(\sqrt{N})$  steps, compared to  $\Omega(N)$  for a classical computer. This quadratic speed-up was then generalized to arbitrary unstructured search problems by a generic amplitude amplification process by Brassard et al. [10].

Grover's algorithm and amplitude amplification are directly applicable to unstructured global search problems, but not to search problems relying on a local realization. Consider we have just inspected one of the  $N$  elements and found that it is not marked, and we want next to inspect another of the  $N$  elements. Many search problems have the localized property that it is less costly to inspect an element that is close to the most recently inspected element, as opposed to inspecting an arbitrary element. Many probabilistic algorithms for such problems use random walks, and the quantum analogue of such are called quantum walks.

Quantum walks have proven very successful in quantum computing, with applications in diverse settings such as communication complexity [1], element distinctness problems [3, 8],

---

\* This work was partially supported by NSERC, the Natural Sciences and Engineering Research Council of Canada and CIFAR, the Canadian Institute for Advanced Research.



testing group commutativity [24], and triangle finding [27, 16]. Excellent surveys on quantum walks, their history and applications, include [2, 19, 32, 36, 31].

The expected number of steps  $H$  required to find a marked element by a random walk is called the *hitting time*. The hitting time depends on the structure being searched as well as the number and locations of the marked elements.

Quantum walks have been studied for many structures, and in particular for the torus. A torus is a graph containing  $N$  vertices laid out in a two-dimensional square structure. It is also called a grid or a two-dimensional lattice. The first work on the torus was by Aaronson and Ambainis [1], who showed that a torus can be searched in  $O(\sqrt{N} \log^2 N)$  steps. Their breakthrough result is remarkably close to the quadratic speed-up that is attainable for unstructured search problems, and it raised the question of determining the limitations of quantum walks in general and on a torus in particular.

For the torus, Ambainis et al. [5] next gave a quantum walk using  $O(\sqrt{N} \log N)$  steps. The question of whether one could find a marked element any faster was solved Tulsi [34] who found a quantum algorithm using  $O(\sqrt{N \log N})$  steps, obtained by attaching an ancilla qubit and thereby modifying the search space. The above results assume the torus contains a single marked element. If there are multiple marked elements, one can probabilistically reduce the number of marked elements, potentially incurring an increased cost, and not what we would naturally expect and desire, a decreased cost.

For general walks, Szegedy [33] showed in an influential paper how to construct a quantum walk from any given symmetric random walk. Szegedy's algorithm detects the presence of a marked element in a number of steps of order  $\sqrt{H}$  which is quadratically smaller than classical hitting time  $H$ . Szegedy's algorithm applies to any number  $M$  of marked elements, but does not necessarily find a marked element. In some cases, it outputs a marked element with success probability no better than if we simply sampled from the stationary distribution.

Magniez et al. [26] next showed how phase estimation can be applied to the larger class of reversible random walk, and gave an algorithm that both detects and finds a marked element. Their algorithm applies to any number of marked elements, but does not guarantee a quadratic speed-up in the hitting time. Magniez et al. [25] gave a quantum algorithm that detects the presence of a marked element for any reversible random walk in  $O(\sqrt{H})$  steps. As Szegedy's algorithm, it applies to any number of marked elements, but it does not necessarily find a marked element. Magniez et al. [25] also gave a quantum algorithm that finds a unique marked element in  $O(\sqrt{H})$  steps for any state-transitive random walk.

Krovi et al. [23] next introduced the novel idea of interpolating walks. Krovi et al. [22] show that interpolated walks can find a marked element for any reversible random walk, even with multiple marked elements. The algorithm does not guarantee a quadratic speed-up when there are multiple marked elements. Dohotaru and Høyer [15] introduced controlled quantum walks and showed that such walks also find a marked element for any reversible random walk, even with multiple marked elements, but again, not quadratically faster when there are multiple marked elements. The quantum algorithms given in both papers [22] and [15] use a number of steps in the order of a quantity called the *extended hitting time*.

The question of finding a marked element in quadratically fewer steps than by a random walk when there are multiple marked elements, has thus remained the main open question.

The torus has continued to be a canonical graph of study. Ambainis and Kokainis [6] show that for the torus, the extended hitting time can be  $\Theta(N)$  while the hitting time is  $O(1)$  when there are multiple marked elements. On the torus, we can find a unique marked element in  $O(\sqrt{N \log N})$  steps with success probability of order  $1/\log N$  by a continuous-time quantum walk [13] and by a coin-less quantum walk [7]. Ambainis et al. [4] show that the algorithm for



the torus in [5] can be modified, yielding a quantum algorithm that uses  $O(\sqrt{N \log N})$  steps and finds a unique marked element with constant probability. Nahimovs and Santos [30] show that the probability the algorithm of [5] finds a marked element can be as small as  $O(1/N)$  when there are two marked elements. Nahimovs and Rivosh [29] show that the locations of multiple marked elements on the torus can significantly impact the hitting time.

In this work, we give a quantum algorithm that finds a marked element quadratically faster than classically, up to a logarithmic factor, on the torus, no matter the number of marked elements. This is the first known quantum algorithm that finds a marked element faster than the square-root of the extended hitting time. For some instances, the extended hitting time is a factor of  $N$  larger than the hitting time.

We also analyze the extended hitting time. We give a new upper bound on the extended hitting time and prove that it is convex in the marked subset, with respect to the stationary distribution. These results are stated as Theorem 2 and Corollary 3 in Section 2. These two results yield in themselves a simplification of known quantum walks that are based on pruning the number of marked elements.

We next define and discuss the torus graph in Section 3. A major obstacle in finding a better quantum algorithm for the torus has been its locality properties. In Section 4, we investigate the locality properties of a random walk on the torus, and we turn these into our advantage, instead of being a disadvantage. We are sculpturing the connectivity. As argued by Meyer and Wong in [28], connectivity in itself is a poor indicator of fast quantum search. The idea of using properties of the underlying graph to direct the quantum walk to specific parts of the search space has been used elsewhere, e.g. by Le Gall in [16] to obtain the best known quantum algorithm for triangle finding.

In Section 5, we give our new quantum algorithm for finding a marked element on the torus when there are multiple marked elements. Our algorithm uses quadratically fewer steps than a random walk, ignoring logarithmic factors.

## 2 Bounds on the extended hitting time

Consider a Markov chain on a discrete finite state space  $X$  of size  $N$ . We represent its transition function as an  $N \times N$  matrix  $P$ . The entries of  $P$  are real and non-negative. Entry  $P_{yx}$  denotes the probability of transitioning from state  $x$  to state  $y$  in one step. The entries in each column sum to one, implying that  $P$  is column-stochastic. We can consider the matrix  $P$  as the adjacency matrix of an underlying directed weighted graph.

We assume that the chain  $P$  is ergodic, which implies that it has a unique stationary distribution  $\pi$  satisfying that  $P\pi = \pi$ . It follows from the Perron–Frobenius theorem that the stationary distribution  $\pi$  has real and positive entries. A Markov chain is *ergodic* if its underlying graph is strongly connected and acyclic.

We also assume that  $P$  is reversible. A Markov chain is *reversible* if  $P_{yx}\pi_x = P_{xy}\pi_y$  for all states  $x, y \in X$  in the state space. This condition expresses that the same amount of probability transition in either direction between any two states  $x$  and  $y$  in the stationary distribution. From now on, we will only consider Markov chains that are both ergodic and reversible, and we will also refer to such chains as random walks. Reversibility permits us to apply spectral analysis, following the seminal work of Szegedy [33].

Let  $\mathcal{M} \subset X$  be the subset of marked states, and let  $U = X \setminus \mathcal{M}$  be the remaining states which are unmarked. We form the *absorbing* walk  $P'$  from  $P$  by modifying all outgoing edges from marked states into self-loops. That is, if  $x \in \mathcal{M}$  is marked, we set  $P'_{xx} = 1$  and  $P'_{yx} = 0$  for all other states  $y \in X \setminus \{x\}$ . We set  $P'_{yx} = P_{yx}$  for all unmarked states  $x \in U$  and all states  $y \in X$ . We will interchangeably refer to states as “elements.”

The main goal of the random walk  $P$  is to find a marked state. The walk starts in a state drawn from the stationary distribution  $\pi$ . We keep applying the transition function until we reach a marked state, at which point the walk halts. The *hitting time* is the expected number of steps it takes for the random walk to find a marked state, and it is denoted by  $\text{HT}(P, \mathcal{M})$ .

We use spectral analysis to study the hitting time of random walks, as in Szegedy [33]. The *discriminant* of any given random walk  $P$  is the matrix  $D(P) = \sqrt{P \circ P^T}$ , where  $T$  denotes matrix transposition, and where the Hadamard product  $\circ$  denotes entry-wise product and the square-root is taken entry-wise. The discriminant is a symmetric real matrix by definition and thus has real eigenvalues.

We use both the discriminant of the walk  $P$  and its absorbing walk  $P'$ . The discriminant  $D(P)$  of  $P$  has real eigenvalues  $1 = \lambda_1 > \lambda_2 \geq \dots \geq \lambda_N > -1$  with corresponding eigenvectors  $|\lambda_1\rangle, |\lambda_2\rangle, \dots, |\lambda_N\rangle$ . The *spectral gap* of  $P$  is  $\delta = 1 - \lambda_2$ .

The discriminant  $D(P')$  of the absorbing walk  $P'$  has  $|\mathcal{M}|$  eigenvectors  $|x\rangle$  with eigenvalue  $+1$ , one for each marked state  $x \in \mathcal{M}$ . The remaining  $N - |\mathcal{M}|$  eigenvectors  $|\lambda'_1\rangle, |\lambda'_2\rangle, \dots, |\lambda'_{N-|\mathcal{M}}\rangle$  have eigenvalues  $1 > \lambda'_1 \geq \lambda'_2 \geq \dots \geq \lambda'_{N-|\mathcal{M}} > -1$  that are strictly less than one in absolute value and they span the unmarked subspace.

The hitting time of  $P$  can then be expressed in terms of the spectra of the discriminant  $D(P')$  for the absorbing walk. Let  $|\pi\rangle = \sum_{x \in X} \sqrt{\pi_x} |x\rangle$  denote the column vector corresponding to the stationary distribution, normalized entry-wise. For any subset  $S \subseteq X$  of elements, let  $\epsilon_S = \sum_{x \in S} \pi_x$  denote the probability that the stationary distribution is in a state in  $S$ . Let  $|S_\pi\rangle = \frac{1}{\sqrt{\epsilon_S}} \sum_{x \in S} \sqrt{\pi_x} |x\rangle$  denote the normalized projection of the stationary distribution  $|\pi\rangle$  onto the subspace spanned by elements in a non-empty subset  $S$ . Let  $|S_\pi\rangle$  be the vector of length zero if subset  $S$  is empty. In particular, we use  $\epsilon_{\mathcal{M}}$  and  $\epsilon_U$ , as well as  $|\mathcal{M}_\pi\rangle$  and  $|U_\pi\rangle$ , to denote the quantities for the marked and unmarked subsets, respectively.

► **Lemma 1** (see e.g. [33, 31, 22]). *The hitting time of a reversible ergodic Markov chain  $P$  with marked elements  $\mathcal{M}$  is*

$$\text{HT}(P, \mathcal{M}) = \sum_{k=1}^{N-|\mathcal{M}|} \frac{|\langle \lambda'_k | U_\pi \rangle|^2}{1 - \lambda'_k}. \quad (1)$$

The hitting time is the expected number of steps of  $P'$  required to reach a marked vertex, starting from a random unmarked vertex, picked according to the stationary distribution  $\pi$ . We define the *effective hitting time*  $\text{HT}_{\text{eff}}(P, \mathcal{M})$  as the number of steps of  $P'$  required to reach a marked vertex with probability at least  $2/3$ , again starting from a random unmarked vertex, picked according to the stationary distribution  $\pi$ . By Markov's inequality, the effective hitting time is at most three times larger than the hitting time.

In his seminal paper, Szegedy [33] proved that we can *detect* whether a marked element exists or not quadratically faster by a quantum algorithm. If there are  $|\mathcal{M}| > 0$  marked elements  $\mathcal{M}$ , it suffices to run Szegedy's quantum algorithm for  $O(\sqrt{\text{HT}_{\text{eff}}(P, \mathcal{M})})$  steps to determine the existence of a marked element with bounded one-sided error [33, 25].

A breakthrough for quantum walks with multiple marked elements was achieved by Krovi et al. [23, 21, 22]. They introduced a walk  $P(s) = (1 - s)P + sP'$  which is an interpolation between the non-absorbing walk  $P$  and the absorbing walk  $P'$ . The walk is parameterized by a quantity  $0 < s < 1$ , which is chosen to be very close to 1, implying that the walk is almost absorbing. They prove that their algorithm both detects and finds a marked element, even when there are multiple marked elements. The limitation is that their quantum walk does not necessarily guarantee a quadratic speedup over a classical walk. To measure the number of steps of their algorithm, they introduce a quantity  $\text{HT}^+(P, \mathcal{M})$  called the *extended hitting*

*time.* Their algorithm takes a number of steps that is of order  $\sqrt{\text{HT}^+(\mathbb{P}, \mathcal{M})}$ , the square-root of the extended hitting time, ignoring logarithmic factors.

Their work raises two main questions. The first question is to determine the extent to which the extended hitting time can exceed the hitting time. The second question is to continue the quest for the discovery of a quantum algorithm that finds a marked element quadratically faster than a random walk when there are multiple marked elements.

Ambainis and Kokainis [6] considered the question of determining the largest possible ratio between the extended hitting time and the hitting time for a natural search space. They show that especially for the torus, the ratio can be exceptionally large by providing an example of a set of marked elements  $\mathcal{M}$  on the torus for which  $\text{HT}(\mathbb{P}, \mathcal{M}) \in O(1)$ , yet  $\text{HT}^+(\mathbb{P}, \mathcal{M}) \in \Theta(N)$ . That is, the hitting time is a constant, yet the extended hitting time is linear in the size of the torus. Searching with multiple marked elements on the torus in the square-root of the extended hitting time can be remarkably slow.

In the case there is a single marked element ( $\mathcal{M} = \{m\}$ ), the extended hitting time is identical to the hitting time, and thus  $\text{HT}^+(\mathbb{P}, \{m\}) = \text{HT}(\mathbb{P}, \{m\})$ . For multiple marked elements, Ambainis and Kokainis [6] proved a general upper bound on the extended hitting time of  $\text{HT}^+(\mathbb{P}, \mathcal{M}) \leq \frac{1}{\epsilon} \frac{1}{\delta}$ , which implies that the extended hitting time can be at most a factor of  $\frac{1}{\delta}$  larger than the hitting time  $\text{HT}(\mathbb{P}, \mathcal{M})$ . For the torus, the spectral gap  $\delta$  is of order  $\frac{1}{N}$ , which is so small that it permits the above ratio of order  $N$  of the extended hitting time over the hitting time.

To derive an efficient quantum algorithm for the torus for multiple marked elements, we first provide a new upper bound on the extended hitting time. We show that the extended hitting time on a marked set  $\mathcal{M}$  is never more than the weighted average of the hitting times of any its constituents.

► **Theorem 2.** *Let  $\mathbb{P}$  be a reversible ergodic random walk with stationary distribution  $\pi$ . Let  $\mathcal{M} = \cup_i \mathcal{M}_i$  be the disjoint union of non-empty subsets  $\mathcal{M}_i$  of marked elements. The extended hitting time on  $\mathcal{M}$  is at most the weighted average of the extended hitting times of its subsets,*

$$\text{HT}^+(\mathbb{P}, \mathcal{M}) \in O\left(\sum_i \frac{\epsilon_i}{\epsilon_{\mathcal{M}}} \text{HT}^+(\mathbb{P}, \mathcal{M}_i)\right).$$

Here  $\epsilon_i = \sum_{x \in \mathcal{M}_i} \pi_x$  is the probability that  $\pi$  is in marked subset  $\mathcal{M}_i$ , and  $\epsilon_{\mathcal{M}} = \sum_{i \in \mathcal{M}} \epsilon_i$  is the total probability that  $\pi$  is in a marked state.

As a corollary, by letting each subset  $\mathcal{M}_i$  be a singleton set, we obtain that the extended hitting time of a set is never more than the worst-case hitting time of its members.

► **Corollary 3.** *The extended hitting time  $\text{HT}^+(\mathbb{P}, \mathcal{M})$  on a marked subset  $\mathcal{M}$  is in the order of the maximum of the hitting times  $\text{HT}(\mathbb{P}, \{m\})$  of its members  $m \in \mathcal{M}$ .*

There are two technical obstacles in analyzing and understanding the extended hitting time. Firstly, it is defined as a limit of the hitting time of the interpolated walk  $\mathbb{P}(s)$  as the parameter  $s$  approaches 1. Secondly, it is expressed in terms of the spectra of the absorbing walk  $\mathbb{P}'$ , and that spectra changes as we change the set of marked elements. Fortunately, we can circumvent both obstacles by applying the following theorem from [15].

► **Theorem 4** ([15]). *For any reversible ergodic random walk  $\mathbb{P}$  with marked elements  $\mathcal{M}$ ,*

$$\text{HT}^+(\mathbb{P}, \mathcal{M}) \in \Theta\left(\frac{1}{\epsilon_{\mathcal{M}}} \mathbb{E}(\mathbb{P}, \mathcal{M}_{\pi})\right). \quad (2)$$

The theorem expresses the extended hitting time as a product of two factors. The first factor is  $\frac{1}{\epsilon_{\mathcal{M}}}$ , the inverse of the probability that the stationary distribution is in a marked state. The second factor is defined below and is a quantity that is expressed in terms of the spectra of  $P$ , the original walk and not the absorbing walk  $P'$ . Theorem 4 permits us to analyze the extended hitting time by analyzing the original walk  $P$ , a task that is often simpler than analyzing the absorbing walk  $P'$ .

► **Definition 5.** For any normalized vector  $|g\rangle$  over the state space of the walk  $P$ , the *escape time* of  $|g\rangle$  is

$$E(P, |g\rangle) = \sum_{k=2}^N \frac{|\langle \lambda_k | g \rangle|^2}{1 - \lambda_k}. \quad (3)$$

For any non-trivial subset  $S \subseteq X$  of elements, define the escape time of  $S$  with respect to  $\pi$  as  $E(P, S_\pi) = E(P, |S_\pi\rangle)$ .

We will often omit the subscript  $\pi$  and simply write  $E(P, S)$  for  $E(P, S_\pi)$ . By definition, the escape time is a weighted average over the reciprocals of all of the spectral gaps  $1 - \lambda_k$  of the original walk  $P$ . It follows the escape time is at most the inverse of the (smallest) gap  $\delta = 1 - \lambda_2$ . Equation 2 then permits us to re-derive that the extended hitting time is at most  $\frac{1}{\epsilon_{\mathcal{M}} \delta}$ , as shown by Ambainis and Kokainis [6]. The escape time is at least 1/2 for any normalized vector  $|g\rangle$  orthogonal to the principal eigenvector  $|\lambda_1\rangle$ , since the denominator in Eq. 3 is upper bounded by 2. We next show that the escape time is at most additive.

► **Lemma 6.** For any two disjoint subsets of elements  $S_1$  and  $S_2$ ,

$$E(P, S_1 \cup S_2) \leq E(P, S_1) + E(P, S_2).$$

**Proof.** Fix a distribution over the vertex set  $V$ , e.g. the stationary distribution  $\pi$ . The lemma holds trivially if  $S_1$  or  $S_2$  is the empty set, and thus assume that both sets are non-empty. Write the normalized state  $|S_1 \cup S_2\rangle = a|S_1\rangle + b|S_2\rangle$  as a linear combination of the normalized and orthogonal elements  $|S_1\rangle$  and  $|S_2\rangle$ . Noting that  $a^2 + b^2 = 1$ , by the Cauchy–Schwarz inequality,  $|\langle \lambda_k | S_1 \cup S_2 \rangle|^2$  is then at most the sum of  $|\langle \lambda_k | S_1 \rangle|^2$  and  $|\langle \lambda_k | S_2 \rangle|^2$ . We can thus upper bound the sum of the terms  $\frac{|\langle \lambda_k | S_1 \cup S_2 \rangle|^2}{1 - \lambda_k}$  by one sum over terms of the form  $\frac{|\langle \lambda_k | S_1 \rangle|^2}{1 - \lambda_k}$ , plus another sum over terms of the form  $\frac{|\langle \lambda_k | S_2 \rangle|^2}{1 - \lambda_k}$ . The former sum is the escape time of  $S_1$ , the latter the escape time of  $S_2$ . ◀

We next use the sub-additivity of the escape time to prove that the extended hitting time on a marked subset  $\mathcal{M}$  is never more than the extended hitting time of any of constituents.

**Proof of Theorem 2.** Let  $\mathcal{M} = \mathcal{M}_1 \cup \mathcal{M}_2$  be a disjoint union of two non-empty subsets of marked elements. Let  $\epsilon_1 = \epsilon_{\mathcal{M}_1}$  be the probability that the stationary distribution  $\pi$  is in a marked state in  $\mathcal{M}_1$ , and let  $\epsilon_2 = \epsilon_{\mathcal{M}_2}$  be defined similarly. Let  $\epsilon_{\mathcal{M}} = \epsilon_1 + \epsilon_2$ .

Using Equation 2 and Lemma 6, and omitting asymptotic tight factors, write

$$\begin{aligned} \text{HT}^+(P, \mathcal{M}) &= \frac{1}{\epsilon_{\mathcal{M}}} E(P, \mathcal{M}) \\ &\leq \frac{1}{\epsilon_{\mathcal{M}}} (E(P, \mathcal{M}_1) + E(P, \mathcal{M}_2)) \\ &= \frac{\epsilon_1}{\epsilon_{\mathcal{M}}} \frac{1}{\epsilon_1} E(P, \mathcal{M}_1) + \frac{\epsilon_2}{\epsilon_{\mathcal{M}}} \frac{1}{\epsilon_2} E(P, \mathcal{M}_2) \\ &= \frac{\epsilon_1}{\epsilon_{\mathcal{M}}} \text{HT}^+(P, \mathcal{M}_1) + \frac{\epsilon_2}{\epsilon_{\mathcal{M}}} \text{HT}^+(P, \mathcal{M}_2). \end{aligned}$$

Theorem 2 follows by linearity. ◀

Corollary 3 has an important and previously unrecognized consequence. Consider we are given some computational problem that has multiple solutions, and assume that we know how to solve the problem when there is a unique solution. Then we may be able to devise a randomized polynomial-time reduction that probabilistically makes all solutions but one into non-solutions, and then find the only remaining solution. Such pruning ideas have been used in e.g. reducing SAT to unique-SAT [35] and finding a marked element on the torus by Aaronson and Ambainis [1]. As stated in e.g. [25], randomized reductions of multiple marked elements to a unique marked element may increase the cost by a poly-logarithmic factor.

Theorem 2 yields an alternative to such reductions. We simply just run either the controlled quantum walk of [15] or the interpolated quantum walk of [22]. Both algorithms take a number of steps in the order of the square-root of the extended hitting time, ignoring logarithmic factors. By Theorem 2, the extended hitting time of a subset  $\mathcal{M}$  is upper bounded by the average of the hitting times of its members, where the average is with respect to the stationary distribution  $\pi$ . Provided we are given an estimate  $\tilde{\epsilon}$  of  $\epsilon_{\mathcal{M}}$  satisfying that  $\frac{2}{3} \leq \frac{\tilde{\epsilon}}{\epsilon_{\mathcal{M}}} \leq \frac{4}{3}$ , then we find a marked element with probability at least  $1/5$ . (Apply e.g. Theorem 7 in [22] with the value  $\epsilon_2 = \frac{1}{100}$  in their proof.)

► **Corollary 7.** *Given a reversible ergodic Markov chain  $P$  with marked elements  $\mathcal{M}$ , and an estimate  $\tilde{\epsilon}$  of  $\epsilon_{\mathcal{M}}$  satisfying that  $\frac{2}{3} \leq \frac{\tilde{\epsilon}}{\epsilon_{\mathcal{M}}} \leq \frac{4}{3}$ , we can find a marked state by a quantum walk with probability at least  $1/5$  using in the order of  $\sqrt{\text{HT}(P, \{m\})}$  steps, where  $m \in \mathcal{M}$  is chosen to maximize the upper bound.*

One advantage of applying an algorithm that runs in the square-root of the extended hitting time, is that no direct pruning is necessary. We do not need to turn marked elements into non-marked elements. It suffices to guess an estimate  $\tilde{\epsilon}$  of the probability  $\epsilon_{\mathcal{M}}$  of measuring a marked state in the stationary distribution. A second advantage is that the extended hitting time of a subset can be significantly less than the average of the hitting times of its members. The bounds in Theorem 2 and Lemma 6 are only upper bounds, not tight bounds. The bounds can not be improved in general as they are tight for some instances. One such example is the case considered by Nahimovs and Rivosh [29] of the torus with multiple marked elements packed as densely as possible into a sub-square.

### 3 The torus graph

We consider walks on two-dimensional square torus graphs. The graph contains  $N = n^2$  vertices organized into  $n$  rows and  $n$  columns. There is one vertex at location  $(r, c)$  for each row  $0 \leq r < n$  and column  $0 \leq c < n$ . The graph is directed and every vertex has in-degree 4 and out-degree 4.

We consider two types of boundary conditions. We define the *torus* graph in the usual way, with vertices along the boundary connecting to vertices on the opposite boundary. A vertex at location  $(r, c)$  is connected to its four neighbors at locations  $(r - 1, c)$ ,  $(r + 1, c)$ ,  $(r, c - 1)$ , and  $(r, c + 1)$ , where the addition is modulo  $n$ .

We define the *grid* graph to have self-loops on vertices on the boundary. A vertex at location  $(r, c)$  has four out-going edges pointing to the locations  $(\max\{r - 1, 0\}, c)$ ,  $(\min\{r + 1, n - 1\}, c)$ ,  $(r, \max\{c - 1, 0\})$ , and  $(r, \min\{c + 1, n - 1\})$ . Every vertex has in-degree 4 and out-degree 4, as for the torus.

Prior to this Section, all of our discussions have been for the torus, not the grid. Our algorithm given in Section 5 uses both the torus and grid graphs. Since one cannot replace a walk on a torus that crosses the boundary by a walk on a grid without potentially incurring

a cost, we need to clearly distinguish between the two graphs. Our algorithm in Section 5 works for both the torus and grid.

We form a random walk  $P_G$  on a graph  $G$  by taking the adjacency matrix  $\text{Adj}(G)$  of  $G$  and normalize its *columns*. For every directed edge  $(u, v) \in G$ , we set entry  $(v, u)$  in  $P_G$  to be the inverse of the out-degree of vertex  $u$ . All other entries of  $P_G$  are zero. Since both the torus and grid are regular graphs  $G$  with out-degree 4, their random walk operators  $P_G = \frac{1}{4}\text{Adj}(G)$  are scaled versions of their adjacency matrices.

Our proposed quantum algorithm for finding a marked state on the torus uses both the torus and grid graphs. Since the torus and grid are so closely related, it seems intuitively obvious that walking on either graphs should have little influence on the complexity of the algorithm. Indeed, the escape times on the torus and grid with  $N = |V|$  vertices of an element  $m \in V$  are both of order  $\log N$ .

► **Fact 8.** *The escape times of an element  $m \in V$  on the torus and grid are both of order  $\log N$ ,*

$$E(P_{\text{torus}}, \{m\}) \in \Theta(\log N) \quad \text{and} \quad E(P_{\text{grid}}, \{m\}) \in \Theta(\log N).$$

The above fact can be derived from known facts that the hitting time of a unique element on the torus and grid are of order  $N \log N$  and then applying Theorem 4. The fact can also be shown directly by first computing the spectra for the torus and grid, as done in e.g. [5] for the torus, and then applying Definition 5.

#### 4 Locality of random walks on the torus

To obtain a faster quantum algorithm, we first need to resolve the main obstacle that a random walk on a torus is localized.

► **Lemma 9.** *The probability that a random walk of  $T$  steps on an infinite line stays within distance  $\lceil 4\sqrt{T} \rceil$  from the initial position is at least  $1 - 1/745$ .*

**Proof.** Consider a walk on a doubly-infinite line. The walk starts in some fixed initial position, and we measure distances from this initial position. The probability that the walk is at distance (strictly) more than  $k$  from the initial position after  $\ell$  steps is at most  $2 \exp(-\frac{k^2}{2\ell})$  by the Azuma–Hoeffding inequality.

The conditional probability that the walk ends at a distance larger than  $k$ , conditioned on that the walk ever reaches distance  $k + 1$ , is at least  $1/2$  by the reflection principle: once the walk reaches distance  $k + 1$ , the walk is equally likely to end on either side of that location.

By Bayes’s rule, the probability that the walk reaches distance  $k + 1$  is then at most  $4 \exp(-\frac{k^2}{2\ell})$  which is at most  $4e^{-8}$  when  $k = \lceil 4\sqrt{T} \rceil$ . Finally,  $4e^{-8}$  is less than  $1/745$ . ◀

Since the grid is the cartesian graph product of two line graphs, we immediately get that a walk on the grid is also locally contained.

► **Lemma 10.** *The probability that a random walk of  $T$  steps on an infinite grid stays within distance  $\lceil 4\sqrt{T} \rceil$  in all four directions from the initial position is at least  $1 - 2/745$ .*

Let us say that a walk is *localized* if it stays within distance  $\lceil 4\sqrt{T} \rceil$  in all four directions from its initial position  $u$ .

This locality property implies that we can substitute the global walk by disjoint local walks. Consider a torus of size  $n \times n$ , and fix an integer  $1 \leq d < n$ . We cut the torus into  $\Theta((\frac{n}{d})^2)$  disjoint graph components by removing edges from the torus. We remove edges that

cross graph cuts so that each resulting component is a sub-grid (without self-loops on the boundary vertices) and so that all components have length and width that are at least  $D$  and at most  $D + 1$ , for some  $d \leq D < 2d$ . We next add self-loops to every vertex on the boundaries of the resulting graph components. The overall effect is that we have modified the  $n \times n$  torus into  $\Theta((\frac{n}{d})^2)$  disjoint grid graphs, each of size roughly  $D \times D$ , by turning edges between adjacent components into self-loops.

Now, consider a random walk of  $T$  steps on the torus of size  $N = n^2$  starting from the stationary distribution  $\pi$ . We set  $d = 2\lceil 4\sqrt{T} \rceil$  and modify the torus into  $\Theta(\frac{N}{T})$  disjoint sub-grids as described above. We next sample one of these  $\Theta(\frac{N}{T})$  sub-grids according to the stationary distribution  $\pi$  for the torus. That is, we sample the sub-grid  $G$  with probability  $\epsilon_G = \sum_{v \in G} \pi_v$ . The next lemma shows that the probability that a random sub-grid  $G$  contains at least one marked vertex is high.

► **Lemma 11.** *If a random walk of  $T$  steps on the torus of size  $N$  finds a marked vertex with probability at least  $p$ , for  $p \geq \frac{1}{74}$ , then the probability  $p_G$  that a random sub-grid, sampled from the  $\Theta(\frac{N}{T})$  sub-grids as described above, contains at least one marked vertex is at least  $\frac{1}{5}p$ .*

**Proof.** We define the probabilities  $p_{ml}$  and  $p_{Gl}$  below and prove the following three inequalities,

$$p - \frac{2}{745} \leq p_{ml} \leq p_{Gl} \leq 4p_G.$$

By these three inequalities, when  $p \geq \frac{1}{74}$  then  $p_G \geq \frac{1}{5}p$ , and the lemma follows.

Sample a random walk  $\omega$  of length  $T$  as follows: Pick a vertex  $u$  on the torus according to the stationary distribution  $\pi$ , and apply the stochastic matrix  $P_{\text{torus}}$  a number of  $T$  times, starting at  $u$ .

Let  $p_{ml}$  be the probability that a sampled walk is localized and visits a marked vertex. Let  $p_{Gl}$  be the probability that a sampled walk is localized and visits a vertex that is in a sub-grid that contains a marked vertex. Clearly,  $p_{ml} \leq p_{Gl}$ , proving the second inequality.

The unconditional probability that the walk  $\omega$  visits a marked vertex is at least  $p$ . The unconditional probability that  $\omega$  is *not* localized is at most  $\frac{2}{745}$  by Lemma 10. The joint probability that  $\omega$  visits a marked vertex and is localized, is then at least  $p - \frac{2}{745}$ , proving the first inequality.

Fix any sub-grid  $G$  and consider the joint probability that the sampled walk  $\omega$  is localized and visits  $G$ . For a localized walk  $\omega$  to visit  $G$ , its starting vertex must be within distance  $\lceil 4\sqrt{T} \rceil$  of  $G$ . Since  $G$  has width and length at least  $2\lceil 4\sqrt{T} \rceil$ , the number of such vertices is at most four times the number of vertices in  $G$ . The probability of sampling any of these as the starting vertex  $u$  from the stationary distribution  $\pi$ , which is uniform, is then at most four times the probability of sampling the starting vertex  $u$  from  $G$  itself. This proves the third inequality, and thus also the lemma. ◀

In the next Section, we use this locality property in the design of our quantum algorithm and give an efficient algorithm for the torus with multiple marked elements.

## 5 An efficient quantum algorithm for the torus

An implementation of a random walk is done as follows. We first pick a starting node  $v \in V$  for the random walk. This node is picked according to the stationary distribution  $\pi$ . The cost of generating  $v$  is called the *setup cost* and is denoted by  $S$ . We next apply the absorbing walk  $P'$  for some number  $T$  steps. Each step consists of two parts: We first check whether

the node  $v$  we are currently located at is marked or not. The cost of checking whether a node is marked or not is called the *checking cost* and is denoted by  $C$ . If  $v$  is marked, we halt the algorithm and output  $v$ . If  $v$  is not marked, we next apply the operator  $P$  once. The cost of applying  $P$  is called the *update cost* and is denoted by  $U$ . After repeating these two parts for  $T$  steps, we check whether the final node  $v$  is marked or not. If so, we output  $v$ , and if not, we output “failed search.” A random walk with  $T$  steps has total cost  $S + (T + 1)C + TU$ , which we write as  $S + T(U + C)$  by letting the setup cost include the cost of the first checking cost. The walk outputs a marked element with constant probability when  $T \in \Omega(\text{HT}_{\text{eff}}(P_{\text{torus}}, \mathcal{M}))$ .

A quantum walk is implemented similarly [33, 32, 31]. We first create some initial state  $|\text{init}\rangle$  from the state  $|\pi\rangle = \sum_{v \in V} \sqrt{\pi_v} |v\rangle$ , where  $\pi$  is the stationary distribution. We next apply some number  $T_q$  steps of the quantum walk, where each step consists of one application of each of two quantum operators, one denoted  $\text{Ref}(\mathcal{M})$  and one denoted  $W(P)$ , corresponding to the checking and update operators applied in a random walk. The algorithm stops after  $T_q$  steps in some final state. The costs of these three operators are also denoted  $S$ ,  $C$ , and  $U$ , and are in general comparable in cost to the corresponding operators for the random walk. The quantum algorithm has total cost  $S + T_q(U + C)$ .

A measurement of the final state of the quantum walk will not necessarily produce a marked state with constant probability. Had that been the case, then we would have had a quantum algorithm that *finds* a marked element in cost  $S + T_q(U + C)$ . Instead, the quantum walk evolves the initial state  $|\text{init}\rangle$  away from the initial state, and this evolution away from the initial state is sufficient to *detect* that a marked state exists. Szegedy [33] show that after  $T_q$  steps, for some  $T_q \in \Theta(\sqrt{\text{HT}_{\text{eff}}(P, \mathcal{M})})$ , the final state has overlap bounded away from 1 with the initial state. A change by a constant in overlap can be detected by standard techniques such as the swap test [12]. If the swap test shows the final state is different from the initial state, we deduce there is a marked state. We learn that there exists a marked element, but we do *not* necessarily find one. It is possible to efficiently estimate the speed of the change in overlap by applying eigenvalue estimation [20] similar to its uses in e.g. quantum counting [11], phase estimation [14], and quantum walks [26, 25, 22].

► **Theorem 12.** *There exists a quantum algorithm that given a reversible ergodic random walk  $P$  with marked elements  $\mathcal{M} \subset X$ , with probability at least  $2/3$  performs as follows: (1) it outputs an estimate  $\tilde{h} \in O(\text{HT}_{\text{eff}}(P, \mathcal{M}))$  satisfying that if we apply  $P'$  for  $\tilde{h}$  steps starting from the initial distribution  $\pi$ , we find a marked state with probability at least  $3/4$  and (2) it has cost in the order of  $S + \sqrt{\tilde{h}}(U + C)$ .*

Theorem 12 states that there is a quantum algorithm that, with probability at least  $2/3$ , computes an accurate estimate of the effective hitting time efficiently. With complementary probability at most  $1/3$ , this does not happen. We can prevent that the algorithm in Theorem 12 never terminates or terminates after a significant cost. Let  $H_{\text{unique}} = \text{HT}_{\text{eff}}(P_{\text{torus}}, \{m\}) \in \Theta(N \log N)$  be the effective hitting time for the torus when there is a unique marked element. If the algorithm in Theorem 12 has not halted after  $\sqrt{H_{\text{unique}}}$  steps, we halt the algorithm and output  $H_{\text{unique}}$  as our estimate  $\tilde{h}$ .

With this, we can give our quantum algorithm for finding a marked element on the torus. Our algorithm works for multiple marked elements, finds a marked element with probability at least  $1/\log N$ , and has cost in the order of

$$S + \min \{ \sqrt{H \log H}, \sqrt{N \log N} \} (U + C),$$

where  $H = \text{HT}_{\text{eff}}(P_{\text{torus}}, \mathcal{M})$  is the *effective* hitting time on marked subset  $\mathcal{M}$ . This is within a poly-logarithmic factor of being a quadratic speed-up over the cost of a random walk, which has cost the effective hitting time.



► **Theorem 13 (Main).** *There is a quantum algorithm that, given a torus  $P_{\text{torus}}$  with marked vertices  $\mathcal{M} \subset V$ , with probability at least  $2/3$ , outputs a marked element  $m \in \mathcal{M}$  with success probability at least  $\frac{1}{\log N}$  in cost in the order of  $S + \min\{\sqrt{H \log H}, \sqrt{N \log N}\}(U + C)$ , where  $H = \text{HT}_{\text{eff}}(P_{\text{torus}}, \mathcal{M})$  is the effective hitting time.*

The input to the algorithm in Theorem 13 is a torus  $P_{\text{torus}}$  of size  $n \times n$  with some subset  $\mathcal{M} \subset V$  of vertices being marked. The algorithm is as follows.

1. Compute an estimate  $\tilde{h} \in O(\text{HT}_{\text{eff}}(P_{\text{torus}}, \mathcal{M}))$  using Theorem 12. If the algorithm in Theorem 12 has not halted after  $\sqrt{H_{\text{unique}}}$  steps, where  $H_{\text{unique}} = \text{HT}_{\text{eff}}(P_{\text{torus}}, \{m\})$ , halt it and use  $H_{\text{unique}}$  as our estimate  $\tilde{h}$ .
2. Set  $d = 2\lceil 4\sqrt{\tilde{h}} \rceil$ . If  $d > n$ , then set  $d = n$ .
3. Divide the  $n \times n$  torus into disjoint sub-grids so that each sub-grid has length and width between  $D$  and  $D + 1$ , for some  $d \leq D < 2d$ .
4. Create the initial state  $|\pi\rangle = \sum_{v \in P_{\text{torus}}} \sqrt{\pi_v} |v\rangle$  over all vertices in the torus corresponding to the stationary distribution  $\pi$  for the torus.
5. For each vertex  $v \in P_{\text{torus}}$ , assign the name of the sub-grid that  $v$  belongs to in an ancilla register,  $\sum_{v \in P_{\text{torus}}} \sqrt{\pi_v} |v\rangle |\text{subgrid}(v)\rangle$ , in superposition.
6. Set  $\tilde{\epsilon} = \frac{1}{2^k}$ , where integer  $k$  is picked uniformly at random satisfying  $1 \leq 2^k < N$ .
7. Run a controlled quantum walk with estimate  $\tilde{\epsilon}$  on each sub-grid for  $\Theta(D\sqrt{\log D})$  steps in superposition over all sub-grids, by conditioning the walk on the name of the sub-grid in the ancilla register.
8. Measure the final state, producing a vertex  $v$  of the torus. Check if  $v$  is marked. If so, output  $v$ . If not, output “unsuccessful search.”

We first prove the correctness of the algorithm. Assume that the first step of the algorithm outputs a suitable estimate for the effective hitting time as given in Theorem 12. This event happens with probability at least  $\frac{2}{3}$ . Then the probability that a random sub-grid contains at least one marked vertex is at least  $\frac{1}{5} \times \frac{3}{4} = \frac{3}{20}$ , by Lemma 11. For each of those sub-grids, by Corollary 7, the controlled quantum walk in step seven finds a marked element with probability at least  $\frac{1}{5}$ , for at least one of the  $\log N$  possible values for  $k$ . Note that in step seven, each of the conditional walks on the sub-grids start the walk on the state corresponding to the stationary distribution for that sub-grid. The entire algorithm thus outputs a marked element with probability at least  $\frac{2}{3} \times \frac{3}{20} \times \frac{1}{5} = \frac{1}{50}$  for at least one of the  $\log N$  possible values for  $k$ .

The cost of the algorithm is easily deduced. With probability at least  $2/3$ , two properties hold: (1) the estimate  $\tilde{h}$  computed in the first step is in the order of the effective hitting time  $\text{HT}_{\text{eff}}(P_{\text{torus}}, \mathcal{M})$ , and (2) the first and seventh steps of the algorithm each uses a number of steps that is in the order of  $S + \sqrt{\tilde{h} \log \tilde{h}}(U + C)$ . Further, in all events, each of the seven steps of the algorithm never uses more than in the order of  $\sqrt{N \log N}(U + C)$  steps of a quantum walk. Theorem 13 follows.

We remark that we can test all  $\log N$  possible values for  $k$  by testing each of them in turn. This will increase the overall cost by a factor of  $\log N$  and lead to an algorithm with constant success probability. By applying amplitude amplification [10], the increase in cost can be improved to being a factor of order  $\sqrt{\log N}$ . By testing each value of  $k$  in increasing order in turn, our algorithm can be made to have the same cost as the best known quantum algorithms when there is a unique marked element.

We remark that in step seven, we need to run a quantum walk that finds a marked element in the sub-grid, even if the sub-grid contains multiple marked elements. The controlled quantum walk of [15] and the interpolated walk of [22] both do so in  $O(D\sqrt{\log D})$  steps,

when provided an estimate of the probability  $\epsilon_{\mathcal{M}}$ . We also remark that one may omit the conditioning of the quantum walk in step seven by measuring the ancilla register containing the name of a sub-grid immediately after step five. Conducting measurements as early as possible in a quantum algorithm is frequently used when no further computations are required. An early example of such is the semi-classical quantum Fourier transform by Griffiths and Niu [17].

## 6 Concluding remarks

We have given an efficient quantum algorithm for a finding a marked element on the torus with probability at least  $1/\log N$  when there are multiple marked elements. Our algorithm has cost in the order of  $S + \sqrt{H_{\text{eff}} \log H_{\text{eff}}}(U + C)$ , where  $H_{\text{eff}} = \text{HT}_{\text{eff}}(\text{P}_{\text{torus}}, \mathcal{M})$  is the number of steps used by the random walk  $\text{P}_{\text{torus}}$  to find any one of the marked elements  $\mathcal{M}$ . This is a quadratic speed-up, up to a poly-logarithmic factor. It is the first known quantum walk that has cost less than the square-root of the extended hitting time. It is, for the torus, an affirmative answer to the main open question in quantum walks whether it is possible to find a marked element efficiently when there are multiple marked elements.

The study of the torus has proven influential for at least two reasons. Firstly, much progress in quantum walks has been initiated by work on the cycle and the torus, and then later generalized to arbitrary graphs. Secondly, the torus is a hard test-case because the ratio between its hitting time and the reciprocal of its  $\epsilon\delta$  bound [33] is large. For a unique marked element, the hitting time is of order  $N \log N$ , and the reciprocal of the  $\epsilon\delta$  bound is of order  $N^2$ , and thus almost quadratically bigger.

In this work, we have proposed to use localization to our advantage in quantum search, and not as an obstacle to be overcome. We show that localization makes quantum search efficient when there are multiple marked elements on the torus.

---

## References

- 1 Scott Aaronson and Andris Ambainis. Quantum search of spatial regions. *Theory of Computing*, 1(4):47–79, 2005. doi:10.4086/toc.2005.v001a004.
- 2 Andris Ambainis. Quantum walks and their algorithmic applications. *International Journal of Quantum Information*, 1(4):507–518, 2003. doi:10.1142/S0219749903000383.
- 3 Andris Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007. doi:10.1137/S0097539705447311.
- 4 Andris Ambainis, Artūrs Bačkurs, Nikolajs Nahimovs, Raitis Ozols, and Alexander Rivosh. Search by quantum walks on two-dimensional grid without amplitude amplification. In *Conference on the Theory of Quantum Computation, Communication and Cryptography*, TQC'12, pages 87–97, 2012. doi:10.1007/978-3-642-35656-8\_7.
- 5 Andris Ambainis, Julia Kempe, and Alexander Rivosh. Coins make quantum walks faster. In *16th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'05, pages 1099–1108, 2005. arXiv:quant-ph/0402107.
- 6 Andris Ambainis and Mārtiņš Kokainis. Analysis of the extended hitting time and its properties, 2015. Poster presented at the 18th Annual Conference on Quantum Information Processing, QIP'15, Sydney, Australia.
- 7 Andris Ambainis, Renato Portugal, and Nikolajs Nahimovs. Spatial search on grids with minimum memory. *Quantum Information & Computation*, 15(13-14):1233–1247, 2015. arXiv:1312.0172.

- 8 Aleksandrs Belovs, Andrew M. Childs, Stacey Jeffery, Robin Kothari, and Frédéric Magniez. Time-efficient quantum walks for 3-distinctness. In *40th International Colloquium on Automata, Languages, and Programming*, ICALP'13, pages 105–122, 2013. doi:10.1007/978-3-642-39206-1\_10.
- 9 Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, 1997. doi:10.1137/S0097539796300933.
- 10 Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. In Samuel J. Lomonaco, Jr. and Howard E. Brandt, editors, *Quantum Computation and Quantum Information*, volume 305 of *AMS Contemporary Mathematics*, pages 53–74. American Mathematical Society, 2002. arXiv:quant-ph/0005055.
- 11 Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum counting. In *25th International Colloquium on Automata, Languages, and Programming*, ICALP'98, pages 820–831, 1998. doi:10.1007/BFb0055105.
- 12 Harry Buhrman, Richard Cleve, John Watrous, and Ronald de Wolf. Quantum fingerprinting. *Physical Review Letters*, 87(16):167902, 2001. doi:10.1103/PhysRevLett.87.167902.
- 13 Andrew M. Childs and Yimin Ge. Spatial search by continuous-time quantum walks on crystal lattices. *Physical Review A: General Physics*, 89:052337, 2014. doi:10.1103/PhysRevA.89.052337.
- 14 Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. Quantum algorithms revisited. *Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 454(1969):339–354, 1998. doi:10.1098/rspa.1998.0164.
- 15 Cătălin Dohotaru and Peter Høyer. Controlled quantum amplification, 2016. Manuscript. To be presented at the 20th Annual Conference on Quantum Information Processing, QIP'17, Seattle, USA.
- 16 François Le Gall. Improved quantum algorithm for triangle finding via combinatorial arguments. In *55th IEEE Symposium on Foundations of Computer Science*, FOCS'14, pages 216–225, 2014. doi:10.1109/FOCS.2014.31.
- 17 Robert B. Griffiths and Chi-Sheng Niu. Semiclassical Fourier transform for quantum computation. *Physical Review Letters*, 76:3228–3231, 1996. doi:10.1103/PhysRevLett.76.3228.
- 18 Lov K. Grover. A fast quantum mechanical algorithm for database search. In *28th Annual ACM Symposium on Theory of Computing*, STOC'96, pages 212–219, 1996. doi:10.1145/237814.237866.
- 19 Julia Kempe. Quantum random walks: An introductory overview. *Contemporary Physics*, 44(4):307–327, 2003. doi:10.1080/00107151031000110776.
- 20 Alexei Yu. Kitaev. Quantum measurements and the abelian stabilizer problem, 1995. arXiv:quant-ph/9511026.
- 21 Hari Krovi, Frédéric Magniez, Māris Ozols, and Jérémie Roland. Finding is as easy as detecting for quantum walks. In *37th International Colloquium on Automata, Languages, and Programming*, ICALP'10, pages 540–551, 2010. doi:10.1007/978-3-642-14165-2\_46.
- 22 Hari Krovi, Frédéric Magniez, Māris Ozols, and Jérémie Roland. Quantum walks can find a marked element on any graph. *Algorithmica*, 74(2):851–907, 2016. doi:10.1007/s00453-015-9979-8.
- 23 Hari Krovi, Māris Ozols, and Jérémie Roland. Adiabatic condition and the quantum hitting time of Markov chains. *Physical Review A: General Physics*, 82(2):022333, 2010. doi:10.1103/PhysRevA.82.022333.
- 24 Frédéric Magniez and Ashwin Nayak. Quantum complexity of testing group commutativity. *Algorithmica*, 48(3):221–232, 2007. doi:10.1007/s00453-007-0057-8.

- 25 Frédéric Magniez, Ashwin Nayak, Peter C. Richter, and Miklos Santha. On the hitting times of quantum versus random walks. *Algorithmica*, 63(1):91–116, 2012. doi:10.1007/s00453-011-9521-6.
- 26 Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011. doi:10.1137/090745854.
- 27 Frédéric Magniez, Miklos Santha, and Mario Szegedy. Quantum algorithms for the triangle problem. *SIAM Journal on Computing*, 37(2):413–424, 2007. doi:10.1137/050643684.
- 28 David A. Meyer and Thomas G. Wong. Connectivity is a poor indicator of fast quantum search. *Physical Review Letters*, 114:110503, 2015. doi:10.1103/PhysRevLett.114.110503.
- 29 Nikolajs Nahimovs and Alexander Rivosh. Quantum walks on two-dimensional grids with multiple marked locations. In *42nd International Conference on Current Trends in Theory and Practice of Computer Science*, pages 381–391, 2016. doi:10.1007/978-3-662-49192-8\_31.
- 30 Nikolajs Nahimovs and Raqueline A. M. Santos. Adjacent vertices can be hard to find by quantum walks, May 2016. arXiv:1605.05598.
- 31 Ashwin Nayak, Peter C. Richter, and Mario Szegedy. Quantum analogues of Markov chains. In *Encyclopedia of Algorithms*. Springer Berlin Heidelberg, 2014. doi:10.1007/978-3-642-27848-8\_302-2.
- 32 Miklos Santha. Quantum walk based search algorithms. In *International Conference on Theory and Applications of Models of Computation*, TAMC’08, pages 31–46, 2008. doi:10.1007/978-3-540-79228-4\_3.
- 33 Mario Szegedy. Quantum speed-up of Markov chain based algorithms. In *45th IEEE Symposium on Foundations of Computer Science*, FOCS’04, pages 32–41, 2004. doi:10.1109/FOCS.2004.53.
- 34 Avatar Tulsi. Faster quantum walk algorithm for the two dimensional spatial search. *Physical Review A: General Physics*, 78(4):012310, 2008. doi:10.1103/PhysRevA.78.012310.
- 35 Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986. doi:10.1016/0304-3975(86)90135-0.
- 36 Salvador E. Venegas-Andraca. Quantum walks: A comprehensive review. *Quantum Information Processing*, 11(5):1015–1106, 2012. doi:10.1007/s11128-012-0432-5.

# On OBDD-Based Algorithms and Proof Systems That Dynamically Change Order of Variables\*

Dmitry Itsykson<sup>1</sup>, Alexander Knop<sup>2</sup>, Andrey Romashchenko<sup>†3</sup>, and Dmitry Sokolov<sup>4</sup>

- 1 St. Petersburg Department of V. A. Steklov Institute of Mathematics of the Russian Academy of Sciences, St. Petersburg, Russia  
dmitrits@pdmi.ras.ru
- 2 St. Petersburg Department of V. A. Steklov Institute of Mathematics of the Russian Academy of Sciences, St. Petersburg, Russia  
aaknop@gmail.com
- 3 LIRMM, University Montpellier and CNRS, Montpellier, France  
andrei.romashchenko@lirmm.fr
- 4 St. Petersburg Department of V. A. Steklov Institute of Mathematics of the Russian Academy of Sciences, St. Petersburg, Russia  
sokolov.dmt@gmail.com

---

## Abstract

In 2004 Atserias, Kolaitis and Vardi proposed OBDD-based propositional proof systems that prove unsatisfiability of a CNF formula by deduction of identically false OBDD from OBDDs representing clauses of the initial formula. All OBDDs in such proofs have the same order of variables. We initiate the study of OBDD based proof systems that additionally contain a rule that allows to change the order in OBDDs. At first we consider a proof system  $\text{OBDD}(\wedge, \text{reordering})$  that uses the conjunction (join) rule and the rule that allows to change the order. We exponentially separate this proof system from  $\text{OBDD}(\wedge)$ -proof system that uses only the conjunction rule. We prove two exponential lower bounds on the size of  $\text{OBDD}(\wedge, \text{reordering})$ -refutations of Tseitin formulas and the pigeonhole principle. The first lower bound was previously unknown even for  $\text{OBDD}(\wedge)$ -proofs and the second one extends the result of Tveretina et al. from  $\text{OBDD}(\wedge)$  to  $\text{OBDD}(\wedge, \text{reordering})$ .

In 2004 Pan and Vardi proposed an approach to the propositional satisfiability problem based on OBDDs and symbolic quantifier elimination (we denote algorithms based on this approach as  $\text{OBDD}(\wedge, \exists)$ -algorithms). We notice that there exists an  $\text{OBDD}(\wedge, \exists)$ -algorithm that solves satisfiable and unsatisfiable Tseitin formulas in polynomial time. In contrast, we show that there exist formulas representing systems of linear equations over  $\mathbb{F}_2$  that are hard for  $\text{OBDD}(\wedge, \exists, \text{reordering})$ -algorithms. Our hard instances are satisfiable formulas representing systems of linear equations over  $\mathbb{F}_2$  that correspond to some checksum matrices of error correcting codes.

**1998 ACM Subject Classification** F.2.2 [Nonnumerical Algorithms and Problems] Complexity of proof procedures, F.4.1 [Mathematical Logic] Proof Theory

**Keywords and phrases** Proof complexity, OBDD, error-correcting codes, Tseitin formulas, expanders

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.43

---

\* The research presented in Sections 3, 4 and 5 was supported by Russian Science Foundation (project 16-11-10123). The work of the third author on the research presented in Section 6 was partially supported by RFBR grant 16-01-00362.

† On leave from IITP RAS.



## 1 Introduction

An ordered binary decision diagram (OBDD) is a way to represent Boolean functions [3]. A Boolean function is represented as a branching program with two sinks such that on every path from the source to a sink variables appear in the same order. This restriction on the order of variables allows to handle the diagrams (compute binary Boolean operations on diagrams, compute projections, or test satisfiability) very efficiently.

Atserias, Kolaitis and Vardi [1] proposed an OBDD-based proof system. This system is meant to prove that a given CNF formula is unsatisfiable. For some order of variables  $\pi$  we represent clauses of the input formula as a  $\pi$ -ordered BDD; we may derive a new OBDD applying either the *conjunction* rule or the *weakening* rule. (The authors of [1] supplied the system with one more rule – the *projection*, which derives  $\exists xD$  from  $D$ ; we consider this rule as a special case of the weakening rule, so we do not need to allow it explicitly). A *proof* in this system is a derivation of an OBDD that represents the constant false function. We refer to this proof system as the OBDD( $\wedge$ , weakening)-proof system. The OBDD( $\wedge$ , weakening)-proof system simulates Cutting Plane with unary coefficients and thus it is stronger than Resolution. This proof system provides also short refutations for the formulas that represent unsatisfiable systems of linear equations over  $\mathbb{F}_2$  [1], while linear systems are hard for Resolution. This observation motivates the study of the OBDD-based algorithms (notice that the popular DPLL and CDCL algorithms correspond to tree-like and DAG-like Resolutions).

Several exponential lower bounds are known for different versions of OBDD-proof systems. Segerlind [16] proved an exponential lower bound for the tree-like version of OBDD( $\wedge$ , weakening)-proof system using the communication complexity technique proposed in [2]. Krajíček [10] proved an exponential lower bound for the DAG-like version of it using monotone feasible interpolation. Several papers study the OBDD-based proof system that has only one inference rule – the conjunction rule (we refer to this system as OBDD( $\wedge$ )-proof system). Groote and Zantema [8] showed that Resolution does not simulate OBDD( $\wedge$ ). Tveretina, Sinz and Zantema [18] proved the lower bound  $2^{\Omega(n)}$  for the pigeonhole principle  $\text{PHP}_n^{n+1}$  in the OBDD( $\wedge$ )-proof system and proved that OBDD( $\wedge$ ) does not simulate Resolution. Friedman and Xu [6] proved an exponential lower bound for the complexity of random unsatisfiable 3-CNF formulas in restricted versions of OBDD( $\wedge$ )-proof systems (with a fixed order of the variables) and an exponential lower bound for the running time on random unsatisfiable 3XOR formulas of restricted versions of OBDD( $\wedge$ )-proof systems (with fixed orders of application of rules).

An interesting approach to solving propositional satisfiability was suggested by Pan and Vardi [15]. They proposed an algorithm that chooses some order  $\pi$  on the variables of the input CNF formula  $F$  and creates the current  $\pi$ -ordered BDD  $D$  that initially represents the constant true function, and a set of clauses  $S$  that initially consists of all clauses of the formula  $F$ . Then the algorithm applies the following operations in an arbitrary order:

1. conjunction (or join): choose a clause  $C \in S$ , delete it from  $S$  and replace  $D$  by the  $\pi$ -ordered BDD representing  $C \wedge D$ ;
2. projection (or  $\exists$ -elimination): choose a variable  $x$  that has no occurrence in the clauses from  $S$  and replace  $D$  by the  $\pi$ -ordered BDD representing  $\exists xD$ .

When  $S$  becomes empty, the algorithm stops and reports “unsatisfiable” if  $D$  represents the constant false function and “satisfiable” otherwise. Every particular instance of this algorithm uses its own strategies to choose an order of variables  $\pi$  and an order of application of the operations. We refer to these algorithms as OBDD( $\wedge$ ,  $\exists$ )-algorithms.

The lower bounds for the  $\text{OBDD}(\wedge, \text{weakening})$ -proof systems mentioned above imply the same lower bounds for the  $\text{OBDD}(\wedge, \exists)$ -algorithms.

Pan and Vardi [15] investigated some specific strategies and compared them with DPLL based SAT solvers and compared the strategies with SAT solvers based on OBDDs without quantifier eliminations (we call them  $\text{OBDD}(\wedge)$ -algorithms). Experiments showed in particular that  $\text{OBDD}(\wedge, \exists)$ -algorithms are faster than  $\text{OBDD}(\wedge)$ -algorithms and DPLL based algorithms on many natural formulas.

One of these formulas was  $\text{PHP}_n^{n+1}$ . The result of [4] implies that an  $\text{OBDD}(\wedge, \exists)$ -algorithm can solve  $\text{PHP}_n^{n+1}$  in polynomial time in compare to  $\text{OBDD}(\wedge)$ -algorithms and DPLL based algorithms.

## 1.1 Statement of the problem

It is known that changing the order of the variables in an OBDD can be performed in time polynomial in the sizes of the input and the output. So it seems to be very restrictive to use the same order of variables in all OBDDs in the proof. Hence we propose to supply the proof system with a supplementary rule that dynamically reorders the variables in OBDDs.

In OBDD-proofs, the reordering rule may be applied to an arbitrary OBDD from the proof but the conjunction rule may be applied only to OBDDs with the same order since the verification of the application of the conjunction to OBDDs in different orders is hard (this problem is indeed  $\text{coNP}$ -complete, see [14, Lemma 8.14]).

The first aim of this paper is to prove lower bounds for the OBDD-based algorithms and proof systems that use the reordering rule. The second aim is to show that reordering of the variables is really useful; we give examples of formulas that are hard for the OBDD-based proof systems without reordering and easy with reordering.

## 1.2 Our results

In Section 3 we consider the  $\text{OBDD}(\wedge, \text{reordering})$ -proof system. We prove two exponential lower bounds for the size of a  $\text{OBDD}(\wedge, \text{reordering})$ -derivation of the pigeonhole principle  $\text{PHP}_n^{n+1}$  and Tseitin formulas based on constant-degree algebraic expanders. The lower bound for pigeonhole principle extends the result of Tveretina et al. [18] from  $\text{OBDD}(\wedge)$ -proofs to  $\text{OBDD}(\wedge, \text{reordering})$ -proofs. (Besides, we believe that our argument is simpler than the proof in [18]). The result for Tseitin formulas, to the best of our knowledge, was not known even for the more restrictive  $\text{OBDD}(\wedge)$ -proofs. In both arguments we use the same strategy.

- At first step, we prove an exponential lower bound on the size of the OBDD-representation for an appropriate satisfiable formula. Assume for simplicity that the original unsatisfiable formula is minimally unsatisfiable. Roughly speaking, the satisfiable formula under consideration is equal to the original unsatisfiable formula with one canceled clause. For example, for the pigeonhole principle the appropriate satisfiable formula would be  $\text{PHP}_n^n$ ; for the Tseitin formulas such an appropriate formula is a satisfiable Tseitin formula. This part of the proof is quite cumbersome but it involves only rather elementary techniques of lower bounds for OBDD.
- Consider the last derivation step. It consists in the conjunction for  $F_1$  and  $F_2$  in the same order  $\pi$ . Our goal is to prove that at least one of  $F_1$  and  $F_2$  has an exponential size. Both  $F_1$  and  $F_2$  are satisfiable and they are conjunctions of different sets of clauses from the initial formulas. The idea is to construct partial substitutions  $\rho_1$  and  $\rho_2$  with the same support such that the formula  $F_1|_{\rho_1} \wedge F_2|_{\rho_2}$  is isomorphic to the satisfiable formula from the first step. Then any OBDD representation of  $F_1|_{\rho_1} \wedge F_2|_{\rho_2}$  has exponential size.

Hence, the size of either  $F_1|_{\rho_1}$  or  $F_2|_{\rho_2}$  is large for the order  $\pi$ . Thus, the size of either  $F_1$  or  $F_2$  is large for the order  $\pi$ .

Though our method has limitations, it may be applied to not minimally unsatisfiable formulas as well. Roughly speaking, our proof requires that all minimally unsatisfiable subformulas of the formula have simple structures. For example, our proof for  $\text{PHP}_n^{n+1}$  transfers to  $\text{onto-FPHP}_n^{n+1}$  almost literally since the unique minimal unsatisfiable subformula of  $\text{onto-FPHP}_n^{n+1}$  is exactly  $\text{PHP}_n^{n+1}$ . We do not believe that the same technique applies to a random 3-XOR.

In Section 4 we construct an example of a family of formulas that are easy for the  $\text{OBDD}(\wedge, \text{reordering})$ -proof system but hard for the  $\text{OBDD}(\wedge)$ -proof system.

In Section 5 we study  $\text{OBDD}(\wedge, \exists, \text{reordering})$ -algorithms. At first, we notice that there exists an  $\text{OBDD}(\wedge, \exists)$ -algorithm that solves satisfiable and unsatisfiable Tseitin formulas in polynomial time. In contrast, we show that there exist formulas representing systems of linear equations over  $\mathbb{F}_2$  that are hard for  $\text{OBDD}(\wedge, \exists, \text{reordering})$ -algorithms. More specifically, we describe a randomized construction of a family of satisfiable formulas  $F_n$  on  $n$  variables in  $O(1)$ -CNF such that every  $\text{OBDD}(\wedge, \exists, \text{reordering})$ -algorithm runs at least  $2^{\Omega(n)}$  steps on  $F_n$ .

The plan of the proof is as follows.

- We prove that if  $\mathcal{C} \subseteq \{0, 1\}^n$  is the set of codewords of a list-decodable code that allows to correct  $\frac{2}{3}n$  of erasures, then every OBDD that represents the characteristic function of  $\mathcal{C}$  ( $\chi_{\mathcal{C}}$ ) has a big size (this size proves to be close to the number of all codewords in the code). Moreover, this property holds even for the projections of  $\chi_{\mathcal{C}}$  onto any  $\frac{1}{6}n$  coordinates. Notice that a similar result was known for error-correcting codes with large enough minimal distance (see for example the paper of Duris et al. [5]). Guruswami [9] showed that the codes with large enough minimal distance are erasure list-decodable but the opposite statement does not hold.
- We give a randomized construction of the required linear code. This construction is based on random checksum matrix. We use the codes of Gallager [7] that contain  $O(1)$  ones per row. We prove that such a random code with a high probability enjoys the following expansion property: every  $\frac{1}{6}n$  columns of the matrix contain ones in almost all rows.
- We consider the execution of an  $\text{OBDD}(\wedge, \exists, \text{reordering})$ -algorithm on a CNF formula that corresponds to the CNF representation of the checksum matrix of the code. We study two cases:
  1. the algorithm applies the projection rule less than  $\frac{n}{6}$  times;
  2. the projection rule is applied at least  $\frac{n}{6}$  times.

In the first case, we focus on the OBDD at the end of the execution of the algorithm; its size should be exponential due to the properties of the code. In the second case, we consider the first moment in the computational process when we apply exactly  $\frac{n}{6}$  projection operations. By the expansion property, OBDD  $D$  is a projection of almost the entire formula, thus its size should be close to the size of the OBDD of the characteristic function of the code. That is, the size of  $D$  should be large enough.

As we mentioned above, the previously known lower bounds for tree-like and DAG-like  $\text{OBDD}(\wedge, \text{weakening})$ -proofs imply lower bounds for  $\text{OBDD}(\wedge, \exists)$ -algorithms. So, what is new in our results comparative to the lower bounds proven by Segerlind [17] and Krajíček [10]? First of all, our lower bound also works for the reordering operation. The second advantage of our construction is that we come up with quite a natural class of formulas (our formulas represent linear systems of equations that define some error correcting codes), while the constructions in [17] and [10] seem to be rather artificial. Further, we prove the lower



bound  $2^{\Omega(n)}$  for a formula with  $n$  variables, whereas the previously known lower bounds are of the type  $2^{n^\epsilon}$  (for some  $\epsilon < 1/5$ ). Besides, we proposed a new technique that might be applicable for other classes of formulas.

### 1.3 Open problems

The first open problem is to prove a superpolynomial lower bound for the OBDD( $\wedge$ , weakening, reordering)-proofs. The second open problem is to separate the OBDD( $\wedge$ , weakening, reordering) and the OBDD( $\wedge$ , weakening) proof systems.

## 2 Preliminaries

### 2.1 Ordered binary decision diagrams

An ordered binary decision diagram (OBDD) is a data structure that is used to represent Boolean function [3]. Let  $\Gamma = \{x_1, \dots, x_n\}$  be a set of propositional variables. A binary decision diagram is a directed acyclic graph with one source. Every vertex of the graph is labeled by a variable from  $\Gamma$  or by constants 0 or 1. If a vertex is labeled by a constant, then it is a sink (has out-degree 0). If a vertex is labeled by a variable, then it has exactly two outgoing edges: one edge is labeled by 0 and the other edge is labeled by 1. Every binary decision diagram defines a Boolean function  $\{0, 1\}^n \rightarrow \{0, 1\}$ . The value of the function for given values of  $x_1, \dots, x_n$  is computed as follows: we start a path at the source and on every step we go along the edge that corresponds to the value of the variable at the current vertex. Every such path reaches the vertex labeled by the constant; this constant is the value of the function.

Let  $\pi$  be a permutation of the set  $\{1, \dots, n\}$ . A  $\pi$ -ordered binary decision diagram (BDD) is a binary decision diagram such that on every path from the source to a sink every variable has at most one occurrence and variable  $x_{\pi(i)}$  can not appear before  $x_{\pi(j)}$  if  $i > j$ . An ordered binary decision diagram (OBDD) is a  $\pi$ -ordered binary decision diagram for some permutation  $\pi$ . The size of an OBDD is the number of vertices in it.

OBDDs have the following nice property: for every order of variables every Boolean function has a unique minimal OBDD. For a given order  $\pi$ , the minimal OBDD of a function  $f$  may be constructed in polynomial time from any  $\pi$ -ordered BDD for the same  $f$ . There are also known polynomial-time algorithms that efficiently perform the operations of conjunction, negation, disjunction and projection (operation that maps diagram  $D$  computing Boolean function  $f(x, y_1, \dots, y_n)$  to diagram  $D'$  computing Boolean function  $\exists x f(x, y_1, \dots, y_n)$ ) to  $\pi$ -ordered ODDs [13]. There is an algorithm running in time polynomial in the size of the input and the output that gets as input a  $\pi$ -ordered diagram  $A$ , a permutation  $\rho$  and returns the minimal  $\rho$ -ordered diagram that represents the same function as  $A$  [13].

### 2.2 OBDD-proofs

If  $F$  is a CNF formula, we say that the sequence  $D_1, D_2, \dots, D_t$  is an OBDD-derivation of  $F$  if  $D_t$  is an OBDD that represents the constant false function, and for all  $1 \leq i \leq t$ ,  $D_i$  is an OBDD that represents a clause of  $F$  or can be obtained by one of the following inference rules:

1. (conjunction or join)  $D_i$  is a  $\pi$ -ordered OBDD, that represents  $D_k \wedge D_l$  for  $1 \leq l, k < i$ , where  $D_k, D_l$  have the same order  $\pi$ ;
2. (weakening) there exists a  $j < i$  such that  $D_j$  and  $D_i$  have the same order  $\pi$ , and  $D_j$  semantically implies  $D_i$ , that is every assignment that satisfies  $D_j$  also satisfies  $D_i$ ;
3. (reordering)  $D_i$  is an OBDD that is equivalent to an OBDD  $D_j$  with  $j < i$ .

We consider several different OBDD-proof systems with different sets of allowed rules. When we need to denote some specific proof system, we indicate the rules specific for this system in brackets. For example, the  $\text{OBDD}(\wedge)$ -proof system uses only the conjunction rule and hence we may assume that all OBDDs in a proof have the same order. We use the notation  $\pi\text{-OBDD}(\wedge)$ -proof if all diagrams in this proof have order  $\pi$ .

### 2.3 OBDD-algorithms for SAT

Pan and Vardi [15] proposed for the Boolean satisfiability problem the following family of algorithms based on OBDDs and symbolic quantifier elimination.

The algorithm gets as an input a CNF formula  $F$ , it chooses some order  $\pi$  on the variables and creates a  $\pi$ -ordered OBDD  $D$  (which initially is equal to the constant true function) and a set of clauses  $S$  (which initially consists of all clauses of the formula  $F$ ). While  $S$  is not empty the algorithm applies one of the following three operations:

1. (join or conjunction) delete some clause  $C$  from  $S$  and replace  $D$  by a  $\pi$ -ordered BDD that represents the conjunction  $D \wedge C$ ;
2. (projection) choose a variable  $x$  that has no occurrences in the clauses from  $S$  and replace  $D$  by a  $\pi$ -ordered BDD for the function  $\exists x D$ ;
3. (reordering) choose a new order on variables  $\pi'$  and replace  $D$  by the equivalent  $\pi'$ -ordered diagram. Assign  $\pi := \pi'$ . (Note that Pan and Vardi did not consider this rule in [15]).

After every step of the algorithm, the following invariant is maintained:  $F$  is satisfiable if and only if  $\bigwedge_{C \in S} C \wedge D$  is satisfiable. After the termination of the algorithm the set  $S$  is empty; if the diagram  $D$  has a path from the source to a vertex labeled by 1, then the algorithm returns “Satisfiable” and returns “Unsatisfiable” otherwise.

We refer to the algorithms of this type as  $\text{OBDD}(\wedge, \exists, \text{reordering})$ -algorithms. Besides, we use a similar notation for algorithms that use some of the rules: we just enumerate the used rules in the brackets. For example, the  $\text{OBDD}(\wedge)$ -algorithms use only the conjunction rule and the  $\text{OBDD}(\wedge, \exists)$ -algorithms use only the conjunction and projection rules.

Since join and projection for OBDDs may be performed in polynomial time and reordering may be performed in time polynomial on the sizes of the input and the output, the running time of an  $\text{OBDD}(\wedge, \exists, \text{reordering})$ -algorithm are polynomially related with the total sum of the sizes of all values in the diagram  $D$ . We ignore the time spent on choosing  $\pi$  and other operations with the permutation.

### 2.4 Error-correcting codes

By a *code* we mean a subset of binary strings with a fixed length. A code  $C \subseteq \{0, 1\}^n$  has a relative distance  $\delta$  if for any two codewords  $c_1, c_2 \in C$  the Hamming distance between  $c_1$  and  $c_2$  is at least  $\delta n$ .

A *linear code* is a set of all  $n$ -bits vectors  $x = (x_1 \dots x_n)$  from some linear subspace in  $\mathbb{F}_2^n$ . If  $k$  is the dimension of this space, then the ratio  $k/n$  is called the *rate* of the code.

A linear code can be specified by a system of linear equations. For a code of dimension  $k$  this system should consist of  $m \geq n - k$  linear equations involving  $n$  variables. The set of all solutions of the system should give exactly our code, so the rank of the system must be equal to  $n - k$ . If we require in addition that the equations in the system are linearly independent, then the number of equations is equal to  $m = n - k$ . The matrix of this linear system is called a *checksum matrix* of the code.

For a checksum matrix  $(h_{ij})$  over  $\mathbb{F}_2$  we say that a column  $i$  intersects a row  $j$ , if  $h_{ij} = 1$ . Further, we say that a tuple of columns  $\langle i_1, \dots, i_s \rangle$  intersects some row  $j$  if at least one of the columns  $i_1, \dots, i_s$  intersects row  $j$ .

We say that a code  $C$  recovers a fraction of  $\rho$  erasures by a list of size  $L$  (or  $C$  is  $(\rho, L)$ -erasure list-decodable) if for any  $w \in \{0, 1, ?\}^n$  such that the number of “?” in  $w$  does not exceed  $\rho n$ , there exist at most  $L$  elements in  $C$  that are consistent with  $w$ . A string  $s \in \{0, 1\}^n$  is consistent with  $w$  if for all  $i$ ,  $w_i \in \{0, 1\}$  implies  $s_i = w_i$ .

► **Theorem 1** ([9, Lemma 2]). *If  $C$  is a code with relative distance  $\delta$ , then for every  $\epsilon > 0$  the code  $C$  is  $((2 - \epsilon)\delta, \frac{2}{\epsilon})$ -erasure list-decodable.*

### 3 Lower bounds for OBDD( $\wedge$ , reordering)

#### 3.1 Tseitin formulas

In this Section we prove an exponential lower bound on the size of OBDD( $\wedge$ , reordering)-proofs of Tseitin formulas.

A Tseitin formula  $TS_{G,c}$  is based on an undirected graph  $G(V, E)$  with degree bounded by a constant  $d$ . Every edge  $e \in E$  has the corresponding propositional variable  $p_e$  (in fact variables for loops are not used). There is a function  $c : V \rightarrow \{0, 1\}$ , we call it the labelling function. For every vertex  $v \in V$  we write down a formula in CNF that encodes  $\sum_{u \in V: (u,v) \in E, u \neq v} p_{(u,v)} \equiv c(v) \pmod{2}$ . The conjunction of the formulas described above is called a Tseitin formula. If  $\sum_{v \in U} c(v) = 1$  for some connected component  $U \subseteq V$ , then the Tseitin formula is unsatisfiable. Indeed, if we sum up (modulo 2) all equalities stated in vertices from  $U$  we get  $0 = 1$  since every variable has exactly 2 occurrences. If  $\sum_{v \in U} c(v) = 0$  for every connected component  $U$ , then the Tseitin formula is satisfiable ([19, Lemma 4.1]).

Note that if formulas  $TS_{G,c}$  and  $TS_{G,c'}$  are satisfiable and  $c \neq c'$  then  $TS_{G,c}$  and  $TS_{G,c'}$  are different functions since any satisfying assignment of  $TS_{G,c}$  can not satisfy  $TS_{G,c'}$ .

In the following, we denote the number of connected components of a graph  $G$  by  $\sharp G$ .

► **Theorem 2.** *Let graph  $G$  with vertices  $V$  and edges  $E$  have the following property: for some  $m \in [|E|]$  and  $k > 0$  for all subsets  $E' \subseteq E$  of the size  $m$  the inequality  $\sharp G' + \sharp G'' \leq k$  holds, where  $G'$  and  $G''$  are graphs with vertices  $V$  and edges  $E'$  and  $E \setminus E'$  respectively. If  $TS_{G,c}$  is satisfiable then any OBDD for  $TS_{G,c}$  has at least  $2^{|V|-k}$  vetices on the distance  $m$  from the source.*

**Proof.** Let us fix an order  $\pi$  of the variables of  $TS_{G,c}$  (i.e.  $\pi$  orders the edges of  $G$ ). Let  $E'$  be the set of the first  $m$  edges in this order. We show that there are at least  $2^{|V|-k}$  substitutions to the variables from  $E'$  such that applying each of these substitutions to  $TS_{G,c}$  results in  $2^{|V|-k}$  different functions. It implies that the size of every OBDD representing  $TS_{G,c}$  has at least  $2^{|V|-k}$  vetices on the distance  $m$  from the source, since these substitutions correspond to paths in the OBDD with different endpoints.

Let  $c' : V \rightarrow \{0, 1\}$  be a labeling function that corresponds to a partial substitution with support  $E'$ : in every vertex  $v$ ,  $c(v)$  is the sum modulo 2 of the values of all edges from  $E'$  that are incident to  $v$ . Note that making a partial substitution to a Tseitin formula  $TS_{G,c}$  gives as the resulting formula again a Tseitin formula – some formula  $TS_{G'',c+c'}$ , where  $G''$  is a graph with vertices  $V$  and edges  $E \setminus E'$ , and  $c + c'$  is the sum of the functions  $c$  and  $c'$  modulo 2.

We will prove a lower bound for the number of different  $c'$  such that they can be obtained by a substitution and  $TS_{G'',c+c'}$  is satisfiable. The required properties of  $c'$  can be described

by a system of linear equations with variables  $c'(v)$  for  $v \in V$ : for every connected component  $U$  of graph  $G'$  with vertices  $V$  and edges  $E'$  we put down the equation:  $\sum_{v \in U} c'(v) = 0$  (this subsystem states that  $c'$  can be obtained by a substitution or in other words that  $TS_{G',c'}$  is satisfiable) and for each connected component  $W$  of  $G''$  we put down the equation:  $\sum_{v \in W} c(v) + c'(v) = 0$  (this subsystem corresponds to the satisfiability of  $TS_{G'',c+c'}$ ).

The system has a solution since  $TS_{G,c}$  is satisfiable. There are  $|V|$  variables and at most  $\#G' + \#G'' \leq k$  equations. Hence there is at least  $2^{|V|-k}$  solutions. Different solutions corresponds to different satisfiable formulas  $TS_{G'',c+c'}$  and thus to different functions. ◀

We will apply Theorem 2 to algebraic expanders.

► **Definition 3.** A graph  $G$  with vertices  $V$  and edges  $E$  is an  $(n, d, \alpha)$ -algebraic expander, if  $|V| = n$ , the degree of any vertex in  $V$  equals  $d$  and the absolute value of the second largest eigenvalue of the adjacency matrix of  $G$  is not greater than  $\alpha d$ .

It is well known that for all  $\alpha > 0$  and all large enough constants  $d$  there exists a family  $G_n$  of  $(n, d, \alpha)$ -algebraic expanders. There are explicit constructions such that  $G_n$  can be constructed in  $\text{poly}(n)$  time [12]. Also, it is known that a random  $d$ -regular graph is a good enough expander with high probability.

► **Theorem 4.** Let  $G(V, E)$  be an  $(n, d, \alpha)$ -algebraic expander for  $\alpha < \frac{1}{2}$ . Then for any  $E' \subseteq E$  if  $|E'| = \frac{n}{4d}$ , then  $\#G' + \#G'' \leq n(1 - \epsilon) + 2$ , where  $G'$  is a graph with vertices  $V$  and edges  $E'$ ,  $G''$  is a graph with vertices  $V$  and edges  $E \setminus E'$  and  $\epsilon = \frac{1}{8d(\alpha d + 1)} - \frac{1}{d^2}$ . Note that  $\epsilon > 0$  if  $\alpha < 1/32$  and  $d \geq 4$ .

► **Corollary 5.** If graph  $G$  is an  $(n, d, \alpha)$ -algebraic expander for  $\alpha < \frac{1}{32}$  and  $d \geq 4$ , and a Tseitin formula  $TS_{G,c}$  is satisfiable, then the size of any OBDD for  $TS_{G,c}$  is  $2^{\Omega(n)}$ .

**Proof.** Follows from Theorems 2 and 4. ◀

► **Theorem 6.** Let  $G$  be an  $(n, d, \alpha)$ -algebraic expander with  $d \geq 50$ ,  $\alpha < \frac{1}{32}$ . Then any OBDD( $\wedge$ , reordering)-proof of any unsatisfiable Tseitin formula  $TS_{G,c}$  has the size at least  $2^{\Omega(n)}$ .

**Sketch of the Proof.** We consider the last step of the proof: the conjunction of OBDDs  $F_1$  and  $F_2$  is the identically false function but both  $F_1$  and  $F_2$  are satisfiable. Both  $F_1$  and  $F_2$  are conjunctions of several clauses of  $TS_{G,c}$ . We use that a satisfiable Tseitin formula based on an expander has only exponential sized OBDDs. Moreover, if the underlying graph differs from some expander by  $o(n)$  edges, then any of its OBDD representations has also an exponential size, since the number of connected component of graphs  $G'$  and  $G''$  in Theorem 4 changes by at most  $o(n)$ .

Note that  $F_1$  and  $F_2$  together contains all clauses of  $TS_{G,c}$ . The main case is the following: there are two nonadjacent vertices  $u$  and  $v$  such that  $F_1$  does not contain a clause  $C_u$  that corresponds to the vertex  $u$  and  $F_2$  does not contain a clause  $C_v$  that corresponds to  $v$ . We consider two partial substitutions  $\rho_1$  and  $\rho_2$  that are defined on edges adjacent with  $u$  and  $v$  and on the edges of the shortest path  $p$  between  $u$  and  $v$ . The substitutions  $\rho_1$  and  $\rho_2$  assign opposite values to edges from the path  $p$  and are consistent on all other edges. The substitution  $\rho_1$  satisfies  $C_v$  and refutes  $C_u$  and  $\rho_2$  satisfies  $C_u$  and refutes  $C_v$ .

By the construction  $F_1|_{\rho_1} \wedge F_2|_{\rho_2}$  is the satisfiable Tseitin formula based on the graph that is obtained from  $G$  by deletion of the vertices  $u$  and  $v$  and all edges from the path  $p$  (it is also possible that this formula does not contain some clauses for the vertices from  $p$ ). The size of an OBDD representation of such a formula is exponential since the difference of the

underlying graph from the expander is at most  $o(n)$ . (Note that  $p$  is the shortest path in the expander, thus  $p$  contains at most  $O(\log n)$  edges). Hence we get that either  $F_1$  or  $F_2$  has an exponential size in the given order. ◀

### 3.2 Pigeonhole principle

Let  $m$  and  $n$  be integers and  $p_{i,j}$  be different variables;  $p_{i,j}$  states whether the  $i$ th pigeon is in the  $j$ th hole or not. A formula  $\text{PHP}_n^m$  has two types of clauses. Clauses of the first type (long clauses) states that every pigeon is in at least one hole:  $\bigvee_{j=1}^n p_{i,j}$  for all  $i \in [m]$ . Clauses of the second type (short clauses) states that in every hole there is at most one pigeon:  $\neg p_{i,k} \vee \neg p_{j,k}$  for all  $k \in [n]$  and all  $i \neq j \in [m]$ .

► **Theorem 7.** *Any OBDD( $\wedge$ , reordering)-proof of the pigeonhole principle formula  $\text{PHP}_n^{n+1}$  has size at least  $2^{\Omega(n)}$ .*

## 4 OBDD( $\wedge$ , reordering) is stronger than OBDD( $\wedge$ )

In this section we give an example of a family of unsatisfiable formulas  $\Phi_n$  that have OBDD( $\wedge$ , reordering)-proofs of polynomial size while all OBDD( $\wedge$ )-proofs have size at least  $2^{\Omega(n)}$ .

► **Theorem 8.** *Let  $\Psi_n(x_1, x_2, \dots, x_n)$  be a family of unsatisfiable formulas of size  $\text{poly}(n)$  that satisfies the following conditions:*

- *there exists an order  $\tau$  such that  $\Psi_n$  has  $\tau$ -OBDD( $\wedge$ ) refutation of the size  $\text{poly}(n)$ ;*
- *there exists a polynomial  $p(n)$ , a function  $k : \mathbb{N} \rightarrow \mathbb{N}$  with  $k(n) \leq \log p(n)$  and permutations  $\sigma_1, \sigma_2, \dots, \sigma_{2^{k(n)}} \in S_n$  such that for any permutation  $\pi \in S_n$  there exists  $i \in [2^{k(n)}]$  such that any  $\pi\sigma_i$ -OBDD( $\wedge$ )-proof of  $\Psi_n$  has the size at least  $2^{\Omega(n)}$ .*

Then the formula

$$\Phi_n(w_1, w_2, \dots, w_k, x_1, x_2, \dots, x_n) = \bigwedge_{i=1}^{2^{k(n)}} ((w = i - 1) \rightarrow \Psi(x_{\sigma_i(1)}, x_{\sigma_i(2)}, \dots, x_{\sigma_i(n)}))$$

has an OBDD( $\wedge$ , reordering)-proof of the size  $\text{poly}(n)$ , but any OBDD( $\wedge$ )-proof has the size at least  $2^{\Omega(n)}$ . Here the equality  $w = i - 1$  means that  $w_1 w_2 \dots w_k$  is a binary representation of the integer  $i - 1$ . We assume that  $\Phi_n$  is written in CNF as follows: we add a clause  $\neg(w = i - 1)$  to every clause of  $\Psi(x_{\sigma_i(1)}, x_{\sigma_i(2)}, \dots, x_{\sigma_i(n)})$ .

The similar construction was used by Segerlind [17] in order to show that tree-like OBDD( $\wedge$ , weakening) does not simulate Resolution.

**Sketch of the Proof.** The lower bound. Consider an OBDD( $\wedge$ )-proof  $T$  of the formula  $\Phi_n$ , let  $\tau$  be the order of the variables  $x_1, x_2, \dots, x_n$  that is induced by the order from  $T$ . By the statement of the theorem, there exists  $1 \leq i \leq 2^k$  such that all  $(\tau\sigma_i)$ -OBDD( $\wedge$ )-proofs of  $\Psi$  have at least exponential size. We make a substitution  $w = i - 1$  to the proof  $T$ . This substitution converts the proof of  $\Phi_n$  into a proof of  $\Psi_n$  with the order  $\tau\sigma_i$ . Hence,  $T$  has an exponential size.

The upper bound. Since there exists a polynomial sized OBDD( $\wedge$ )-proof of  $\Psi_n$ , then for all  $i$  there is an order  $\pi_i$  (we may assume after the permutation  $\pi$  the variables  $w$  get the leftmost positions) such that there is a  $\pi_i$ -OBDD( $\wedge$ ) derivation of the diagram representing  $w \neq i - 1$ . From all such diagrams for different  $i$  we may construct a polynomial sized refutation of  $\Phi_n$ , since  $w$  contains only  $O(\log n)$  variables. ◀

Now we construct a family of unsatisfiable formulas  $\Psi_n$  that satisfies the conditions of Theorem 8. We use an argument similar to the proofs of the lower bounds for OBDD( $\wedge$ , reordering)-proofs. At first, we construct a function that has the sizes of OBDD representations in different orders close to the required sizes of proofs for  $\Psi_n$ .

Let  $EQ_n : \{0, 1\}^{2n} \rightarrow \{0, 1\}$  be the equality predicate on pairs of  $n$ -bits strings. We denote it  $EQ_n(x_1, x_2, \dots, x_n, y_1, \dots, y_n)$ ; in this notation the value of  $EQ_n$  is true iff  $x_1x_2 \dots x_n = y_1y_2 \dots y_n$ .

► **Proposition 9.** In the order  $x_1, y_1, x_2, y_2, \dots, x_n, y_n$  the function  $EQ_n$  has an OBDD representation of the size  $3n + 2$ .

**Proof.** The proof can be easily done by induction, using the following equation:  
 $EQ_n(x_1, x_2, \dots, x_n, y_1, \dots, y_n) = (x_1 = y_1) \wedge EQ_{n-1}(x_2, \dots, x_n, y_2, \dots, y_n)$ . ◀

The proof of the following lemma is similar to the proof of the  $\Omega(n)$  lower bound on the best communication complexity of the shifted equality function [11, Example 7.9].

► **Lemma 10.** Let  $\sigma_i$  for  $i \in [n]$  be a cyclic permutation of the variables  $y$  that maps  $y_j$  to  $y_{i+j \bmod n+1}$  for any  $j \in [n]$ . Formally  $\sigma_i(j) = j$  for  $j \in [n]$  and  $\sigma_i(n + j) = n + (i + j - 1 \bmod n) + 1$  for  $j \in [n]$ . Then for any order  $\pi$  on  $2n$  variables there exists  $i \in [n]$  such that every  $\pi\sigma_i$ -OBDD representation of  $EQ_n$  has the size at least  $2^{\Omega(n)}$ .

Now we are ready to construct a formula that may be used in Theorem 8. Consider a formula  $\Psi_n(x, y, z)$  from  $3n + 1$  variables (here  $x, y$  are vectors of  $n$  variables and  $z$  is a vector of  $n + 1$  variables) that is the conjunction of CNF representations of the following conditions:  $x_i = y_i$  for all  $i \in [n]$ ;  $z_0$ ;  $(x_i = y_i) \rightarrow (z_{i-1} \rightarrow z_i)$  for all  $i \in [n]$ ;  $\neg z_n$ . Note that  $\Psi_n(x, y, z)$  is unsatisfiable since we have that  $x_i = y_i$  for all  $i$ ; it implies that  $z_i = 1$  for all  $i$ , but  $z_n$  should be zero. The following statement is straightforward.

► **Proposition 11.**  $\Psi_n$  has a OBDD( $\wedge$ )-proof of polynomial size in the order  $z_0, x_1, y_1, z_1, x_2, y_2, z_2, \dots, x_n, y_n, z_n$ .

► **Lemma 12.** For any order  $\pi$  on the variables  $x, y, z$  the size of a  $\pi$ -OBDD( $\wedge$ )-proof of  $\Psi_n$  is at least  $\frac{1}{10}\sqrt{S}$ , where  $S$  is the size of the shortest  $\pi'$ -OBDD representation of  $EQ_n(x, y)$ , where  $\pi'$  is the order induced by  $\pi$  on the variables  $x, y$ .

► **Theorem 13.** There exists an unsatisfiable CNF formula  $\Phi_n$  of size  $\text{poly}(n)$  such that there exists a polynomial size OBDD( $\wedge$ , reordering)-proof of  $\Phi_n$  but every OBDD( $\wedge$ )-proof of  $\Phi_n$  has the size  $2^{\Omega(n)}$ .

**Proof.** By Lemma 12 for every order  $\pi$  a size of  $\pi$ -OBDD( $\wedge$ )-proof of  $\Psi_n$  is at least  $\frac{1}{10}\sqrt{S}$ , where  $S$  is the size of the shortest  $\pi'$ -OBDD representation of  $EQ_n$  and  $\pi'$  is the order induced by  $\pi$ . By Lemma 10 there exists a family of permutations  $\sigma_i$  of  $[2n]$  for  $i \in [n]$  such that for every order  $\tau$  there exists an  $i \in [n]$  such that the size of any  $\tau\sigma_i$ -OBDD( $\wedge$ )-proof of  $\Psi_n$  is at least  $2^{\Omega(n)}$ . By Proposition 11 there exists a required order  $\tau$  and a  $\tau$ -OBDD( $\wedge$ )-proof of  $\Psi_n$  of size  $\text{poly}(n)$ . Theorem 8 gives a construction of the desired formula  $\Phi_n$ . ◀

## 5 OBDD( $\wedge, \exists$ , reordering)-algorithms

It is known that OBDD( $\wedge, \exists$ )-algorithms can prove  $\text{PHP}_n^{n+1}$  in polynomial time [4]. Now we show that the Tseitin formulas are also easy for OBDD( $\wedge, \exists$ )-algorithms.

► **Proposition 14.** There exists an OBDD( $\wedge, \exists$ )-algorithm that solves Tseitin formulas in polynomial time.

**Sketch of the Proof.** Notice that the projection of two linear equations over the common variable is just the sum of these equations. Since every variable has exactly two occurrences in Tseitin formulas, we can sum up all equations in every connected component. ◀

Now we show that there are satisfiable linear systems over  $\mathbb{F}_2$  that are hard for OBDD ( $\wedge, \exists$ , reordering). At first we notice that every OBDD for a characteristic function of a good enough code has at least exponential size.

► **Theorem 15.** Let  $\mathcal{C} \subseteq \{0, 1\}^n$  be a  $(\frac{1}{2} + \epsilon, L)$ -erasure list decoding code. Then every OBDD representing the characteristic function of  $\mathcal{C}$  has the size at least  $\frac{|\mathcal{C}|}{L^2}$ .

Moreover, for every tuple of  $k \in [2\epsilon n]$  different indices  $i_1, \dots, i_k \in [n]$  every OBDD for the Boolean function  $\exists x_{i_1} \dots \exists x_{i_k} \chi_{\mathcal{C}}(x_1, \dots, x_n)$  has the size at least  $\frac{|\mathcal{C}|}{L^2}$ .

► **Lemma 16.** Let  $A$  be an  $m \times n$  checksum matrix of a  $(\rho, L)$ -erasure list decodable code. The matrix  $A'$  is the result of deleting  $r$  rows from  $A$ . Then  $A'$  is a checksum matrix of a  $(\rho, 2^r L)$ -erasure list-decodable code.

► **Theorem 17.** Let  $\mathcal{C} \subseteq \{0, 1\}^n$  be a linear code with relative distance  $\frac{1}{3}$  such that the checksum matrix  $H$  of the code  $\mathcal{C}$  has the following properties:

- $H$  has size  $\alpha n \times n$ , where  $\alpha \in (0; 1)$  is a constant;
- every row of  $H$  contains at most  $t(n)$  ones, where  $t$  is some function;
- every  $\frac{1}{6}n$  columns of  $H$  intersect (contains ones in) at least  $(\alpha - \delta)n$  rows, where  $\delta \in (0, \frac{1-\alpha}{2})$  is a constant.

Let the formula  $F_n$  be the standard representation of  $H(x) = 0$  as a  $t$ -CNF of size at most  $\alpha n 2^{t(n)-1} t(n)$  (every equation is represented in a straightforward way, without any additional variables). Then every OBDD( $\wedge, \exists$ , reordering)-algorithm runs on the formula  $F_n$  for at least  $2^{\Omega(n)}$  steps.

**Proof.** The code  $\mathcal{C}$  has the relative distance  $\frac{1}{3}$ . Hence,  $\mathcal{C}$  is  $(\frac{7}{12}, 8)$ -erasure list-decodable by Theorem 1, choosing  $\epsilon = \frac{1}{4}$ . We consider the execution of an OBDD( $\wedge, \exists$ )-algorithm on the formula  $F_n$ . We prove that at some moment size of a diagram  $D$  will be at least  $2^{\Omega(n)}$ . Assume that the algorithm during its execution applies the projection operation at least  $k = \frac{1}{6}n$  times. We consider the diagram  $D$  just after the first moment when the algorithm applies the projection operation  $k$  times. Let  $D$  represent a function of type  $\exists y_1, \dots, y_k \phi(x_1, \dots, x_{n-k}, y_1, \dots, y_k)$ , where  $\phi$  is the conjunction of several clauses from  $F_n$ . The projection operation on a variable  $x$  can be applied only if all clauses from  $S$  do not depend on  $x$ . Then all clauses corresponding to the linear equations with the variable  $x$  must be among clauses of  $\phi$ . By the assumption of the theorem any  $k$  columns of  $H$  have ones in at least  $(\alpha - \delta)n$  rows. Thus,  $\phi$  contains all clauses from the representation of  $(\alpha - \delta)n$  equations and possibly several other clauses from other equations.

Lemma 16 implies that  $\phi$  is a characteristic function of a  $(\frac{7}{12}, 8 \cdot 2^{\delta n})$ -erasure list-decodable code. The number of satisfying assignments of  $\phi$  is at least the number of solutions of the system  $Hx = 0$ , hence size of the code defined by  $\phi$  is at least  $2^{(1-\alpha)n}$ . By Theorem 15 size of every OBDD for  $\exists y_1, \dots, y_k \phi(x_1, \dots, x_{n-k}, y_1, \dots, y_k)$  is at least  $2^{(1-\alpha)n} 2^{-2\delta n} \frac{1}{16} > 2^{(1-\alpha-2\delta)n-4}$  for every order of variables.

If the algorithm applies the projection operations less than  $\frac{1}{6}n$  times, the argument is similar; we just need to consider the last diagram  $D$ . ◀

In the next section, we will show that there exist linear codes matching the requirements of Theorem 17. These constructions together with Theorem 17 imply the following result:

► **Corollary 18.** *For all large enough  $n$  there exists a CNF formula of size  $O(n)$  with  $n$  Boolean variables, such that every OBDD( $\wedge, \exists$ , reordering)-algorithm runs on this formula at least  $2^{\Omega(n)}$  steps.*

## 6 Code construction

In this section, we use Low Density Parity Codes (LDPC) of Gallager [7] with quite standard parameters (a constant number of ones in each row and in each column of the checksums matrix). We supplement the usual definition of LDPCs with a rather nonconventional property of uniformity. In what follows we prove that most random Gallager’s codes with suitably chosen parameters enjoy this property.

First of all we recall the classic construction of random LDPC from [7]. Let us fix some integer parameters  $t$ ,  $r$ , and  $n$  (assuming that  $t$  divides  $n$ ). Define the “basic” matrix  $A$  of size  $(n/t) \times n$  as a concatenation of  $t$  copies of the identity matrix  $(n/t) \times (n/t)$ .

Notice that each column of  $A$  contains one non-zero element; in each row of  $A$  there are exactly  $t$  non-zero elements. Further, we consider the family of all matrices  $H$  of size  $(rn/t) \times n$  that consist of  $r$  horizontal “blocks” of size  $(n/t) \times n$ , where each block is obtained as a permutation of columns of  $A$ . It is easy to see that in each column of this matrix there are  $r$  ones, and in each row there are exactly  $t$  ones. We introduce the uniform distribution on all matrices of this type. We can interpret these matrices  $H$  as checksums matrices of some linear codes. Gallager proved that most of these codes have rather large minimal distance.

► **Proposition 19** (see [7]). Most (say, at least 90%) of matrices in Gallager’s family define a linear code with parameters approaching Gilbert–Varshamov bound. More precisely, for every  $\delta \in (0, \frac{1}{2})$  and for every  $t$  there exists  $r = r(t)$  such that for large enough  $n$  most matrices from the defined family have the minimal distance  $\geq \delta n$ . Moreover, the ratio  $r(t)/t$  approaches  $h(2\delta)$  as  $t$  goes to infinity, where  $h(x) = -x \log x - (1-x) \log(1-x)$  (the binary entropy). This means that the rate of the code can be made arbitrarily close to  $1 - h(2\delta)$  (i.e., to the Gilbert–Varshamov bound).

A family of linear codes defined above can be specified by parameters  $r, t, n$ . However, it is more convenient to specify these codes by another triple of numbers – by  $(\delta, t, n)$  (assuming that the value  $r = r(\delta, t, n)$  is defined implicitly as the minimal integer such that most codes of the family have the minimal distance greater than  $\delta n$ ). Now we can state the main technical lemma of this section.

► **Lemma 20.** *For all  $\beta \in (0, 1)$ ,  $\gamma < 1$ , and  $\delta \in (0, \frac{1}{2})$ , for all large enough  $t$  most (say, at least 90%) of linear codes from Gallager’s family with parameters  $(\delta, t, n)$  satisfy the following property: every  $\beta n$  columns in the checksum matrix of the code intersect at least a fraction  $\gamma$  of all rows of the matrix.*

► **Corollary 21.** *For the distribution defined above, the system of linear equations  $Hx = 0$  with probability close to 1 can be represented as a CNF of size  $O(n)$ .*

Thus, we obtain Corollary 18 for CNF of size  $O(n)$  (with  $n$  Boolean variables). In other words, for every  $N$  there exists a CNF formula of size  $N$  such that every OBDD( $\wedge, \exists$ , reordering)-algorithm runs on this formula at least  $2^{\Omega(N)}$  steps.



**Acknowledgements.** The authors are grateful to Sam Buss for fruitful discussions and to the anonymous reviewers for useful comments.

---

**References**

---

- 1 Albert Atserias, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint propagation as a proof system. In Mark Wallace, editor, *Principles and Practice of Constraint Programming – CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 77–91. Springer, 2004. doi:10.1007/978-3-540-30201-8\_9.
- 2 Paul Beame, Toniann Pitassi, and Nathan Segerlind. Lower Bounds for Lovász–Schrijver Systems and Beyond Follow from Multiparty Communication Complexity. *SIAM J. Comput.*, 37(3):845–869, 2007. doi:10.1137/060654645.
- 3 Randal E. Bryant. Symbolic manipulation of boolean functions using a graphical representation. In *Proceedings of the 22nd ACM/IEEE conference on Design automation, DAC 1985, Las Vegas, Nevada, USA, 1985.*, pages 688–694, 1985. doi:10.1145/317825.317964.
- 4 Wěi Chén and Wenhui Zhang. A direct construction of polynomial-size OBDD proof of pigeon hole problem. *Information Processing Letters*, 109(10):472–477, 2009. doi:10.1016/j.ipl.2009.01.006.
- 5 Pavol Duris, Juraj Hromkovic, Stasys Jukna, Martin Sauerhoff, and Georg Schnitger. On multi-partition communication complexity. *Inf. Comput.*, 194(1):49–75, 2004. doi:10.1016/j.ic.2004.05.002.
- 6 Luke Friedman and Yixin Xu. Exponential Lower Bounds for Refuting Random Formulas Using Ordered Binary Decision Diagrams. In Andrei A. Bulatov and Arseny M. Shur, editors, *CSR 2013*, volume 7913 of *Lecture Notes in Computer Science*, pages 127–138. Springer, 2013. doi:10.1007/978-3-642-38536-0\_11.
- 7 Robert G. Gallager. Low-density parity-check codes. *Information Theory, IRE Transactions on*, 8(1):21–28, 1962.
- 8 Jan Friso Groote and Hans Zantema. Resolution and binary decision diagrams cannot simulate each other polynomially. *Discrete Applied Mathematics*, 130(2):157–171, 2003. doi:10.1016/S0166-218X(02)00403-1.
- 9 Venkatesan Guruswami. List decoding from erasures: bounds and code constructions. *Information Theory, IEEE Transactions on*, 49(11):2826–2833, 2003. doi:10.1109/tit.2003.815776.
- 10 Jan Krajíček. An exponential lower bound for a constraint propagation proof system based on ordered binary decision diagrams. *Journal of Symbolic Logic*, 73(1):227–237, 2008. doi:10.2178/jsl/1208358751.
- 11 Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.
- 12 A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.
- 13 Christoph Meinel and Anna Slobodova. On the complexity of Constructing Optimal Ordered Binary Decision Diagrams. *Proceedings of Mathematical Foundations of Computer Science*, 841:515–524, 1994.
- 14 Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design: OBDD – Foundations and Applications*. Springer, 1998.
- 15 Guoqiang Pan and Moshe Y. Vardi. Search vs. Symbolic Techniques in Satisfiability Solving. *7th International Conference on Theory and Applications of Satisfiability Testing, SAT 2004, Revised Selected Papers*, 3542:235–250, 2005. doi:10.1007/11527695\_19.
- 16 Nathan Segerlind. Nearly-Exponential Size Lower Bounds for Symbolic Quantifier Elimination Algorithms and OBDD-Based Proofs of Unsatisfiability. *CoRR*, abs/cs/07040, 2007. arXiv:0701054.

- 17 Nathan Segerlind. On the relative efficiency of resolution-like proofs and ordered binary decision diagram proofs. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity, CCC 2008, 23-26 June 2008, College Park, Maryland, USA*, pages 100–111, 2008. doi:10.1109/CCC.2008.34.
- 18 Olga Tveretina, Carsten Sinz, and Hans Zantema. An Exponential Lower Bound on OBDD Refutations for Pigeonhole Formulas. *Proceedings Fourth Athens Colloquium on Algorithms and Complexity*, 4(Acac):13–21, 2009. arXiv:0909.5038, doi:10.4204/EPTCS.4.2.
- 19 Alasdair Urquhart. Hard Examples for Resolution. *JACM*, 34(1):209–219, 1987. doi:10.1145/7531.8928.

# Multiple Random Walks on Paths and Grids\*

Andrej Ivašković<sup>1</sup>, Adrian Kosowski<sup>2</sup>, Dominik Pajak<sup>3</sup>, and Thomas Sauerwald<sup>4</sup>

1 University of Cambridge, Computer Laboratory, Cambridge, UK

2 INRIA Paris, Paris, France

3 Wrocław University of Science and Technology, Wrocław, Poland

4 University of Cambridge, Computer Laboratory, Cambridge, UK

---

## Abstract

We derive several new results on multiple random walks on “low dimensional” graphs.

First, inspired by an example of a *weighted* random walk on a path of three vertices given by Efremenko and Reingold [7], we prove the following dichotomy: as the path length  $n$  tends to infinity, we have a super-linear speed-up w.r.t. the cover time if and only if the number of walks  $k$  is equal to 2. An important ingredient of our proofs is the use of a continuous-time analogue of multiple random walks, which might be of independent interest. Finally, we also present the first tight bounds on the speed-up of the cover time for any  $d$ -dimensional grid with  $d \geq 2$  being an arbitrary constant, and reveal a sharp transition between linear and logarithmic speed-up.

**1998 ACM Subject Classification** G.3 Probability and Statistics

**Keywords and phrases** random walks, randomized algorithms, parallel computing

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.44

## 1 Introduction

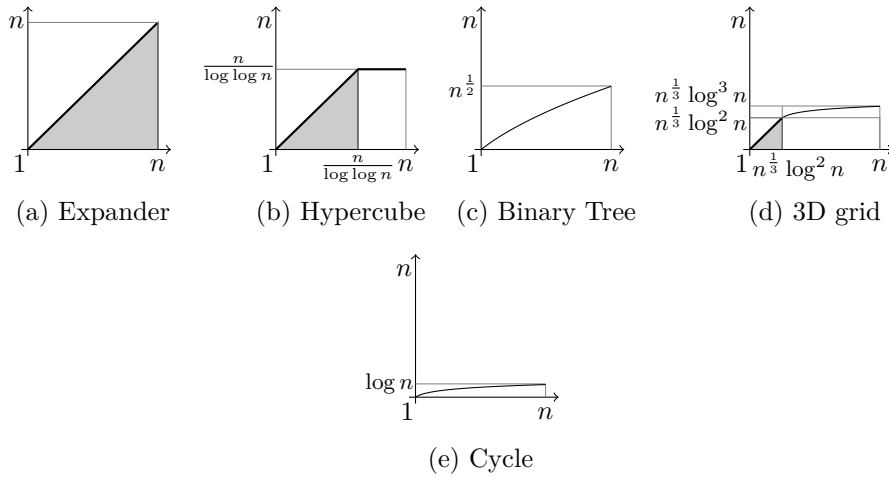
Markov chains are a family of stochastic processes first considered by Markov in 1906. More than a century later, Markov chains are an ubiquitous tool in many areas, including computer science, mathematics, physics and many others (for more background on Markov chains we refer to the textbook [13]). Random walks on graphs are a fundamental instance of a Markov chain, where a particle starts at a given vertex, and at each time-step  $1, 2, \dots$  moves to a random adjacent vertex. One fundamental quantity of random walks is the time it takes to visit all vertices, known as the *cover time*.

In computer science, random walks, or slightly more general, Markov chains, have been instrumental in various breakthrough results. In complexity theory for instance, random walks have played a key role in understanding the space-complexity of algorithms, amplifying the success probability of randomized algorithms and derandomizing algorithms (pseudorandomness). Many sampling tasks in areas like streaming algorithms or sub-linear algorithms can be only solved via random walks. In distributed computing applications, *multiple* random walks are used in protocols for dynamic and anonymous networks, which take advantage of the Markovian property and location-independence of the random walk process, as well as the benefits of random walk parallelization. Such applications include the fundamental tasks of graph exploration with a team of agents [1, 11, 5] and of spreading

---

\* The first and the fourth author were supported by the ERC Starting grant “Dynamic March” funded by the European Research Council. The second and the third author were supported by the National Science Centre, Poland – grant number 2015/17/B/ST6/01897.





■ **Figure 1** Illustration of the asymptotic values of the speed-up  $S_{cov}^{(k)}$  of the cover time of  $k$  independent multiple random walks with identical start vertices over the cover time of a single random walk.

information in a network using multiple tokens [17]. Concurrent random walks are also used in token circulation routines, ensuring mutual exclusion properties in self-stabilizing computation [9], sampling applications [18] and population protocols [16].

Although many of these applications involve multiple and possibly concurrent random walks, comparatively little is known about the theory of multiple random walks. In particular, one might be interested in the so-called **speed-up** that quantifies “how much are  $k$  random walks faster than one”. In one of the first studies, Broder et al. [3] derived bounds on cover time of  $k$  random walks, each starting from a stationary distribution. Their main motivation was to derive time-space tradeoffs for the  $s$ - $t$ -connectivity problem on undirected graphs. However, being able to launch several random walks at randomly chosen vertices might not be always possible in large sampling applications such as crawling massive networks like Facebook or Twitter. Also from a theoretical perspective, it might be natural to study a “worst-case” version of the speed-up.

This might have been one reason why in 2006, Alon et al. [1] introduced the following intriguing mathematical notion of the speed-up: the speed-up  $S_{cov}^{(k)}$  is the ratio of the single worst-case cover time of one random walk to the worst-case cover time of  $k$  independent and concurrent random walks (see Section 2 for a formal definition and further background on (concurrent) random walks). This definition of speed-up is very natural and captures the “parallelism” of random walks in the time they explore a network.

Unfortunately, determining speed-up  $S_{cov}^{(k)}$ , or equivalently determining the  $k$ -walk cover time exactly turns out to be surprisingly difficult, even for basic topologies [1, 7]. Nonetheless, Alon et al. [1] and subsequent studies [7, 19, 8] have revealed a surprisingly broad spectrum of how the speed-up behaves, sometimes even on the same graph, as illustrated in Figure 1.

Based on the two extreme cases of an expander and a cycle, Alon et al. [1] conjectured that the speed-up is always at most  $O(k)$  and at least  $\Omega(\log k)$ . While some progress has been made towards establishing weaker version of these conjectures (cf. [8, 7]), it is still unknown whether they hold in full generality.

From a mathematical perspective, multiple random walks pose a number of intricate challenges, since unlike in case of a single random walk, we often have to look into complex



■ **Figure 2** The Markov chain given by Efremenko and Reingold [7]. The cover time for the single random walk equals  $\frac{5}{1-\alpha}$ , while the cover time for the two random walks starting from any endpoint is  $\frac{2.25}{1-\alpha} + o(1/(1-\alpha))$ , as  $\alpha \rightarrow 1$ .

distributions such as the hitting time distribution (as opposed to simply estimating the expectation in case of a single random walk). In addition to that, we often have to work with “short” random walks, meaning that the random walks’ distribution may be still far from the stationary distribution. This challenge has been identified in several previous works, including in sampling [14] and property-testing [6]. It was also mentioned by Efremenko and Reingold [7], who further highlighted the general complexity of multiple random walks by the following remarkable example.

Take a path with just three vertices (see also Figure 2 on the next page), and consider the speed-up of two random walks. The worst-case start vertex for the single random walk is the middle vertex, while it is not difficult to compute that the worst-case start configuration for two random walks is when they start from the same endpoint. Taking the ratio of the single walk to the 2-walk cover time, one obtains a speed-up of 2.25! This intriguing example serves as a motivation of our study here, as it leads to the following questions:

1. Can we find other, possibly larger, graphs for which the speed-up is super-linear?
2. What happens in case of the path, if  $k$  or  $n$  grows?
3. One subtle detail in the example (Figure 2) are loop probabilities that need to be sufficiently large. Are the loop probabilities necessary, and how do the loop probabilities impact the speed-up?

In this work we will shed some light on the questions above, in particular, the second and third one. Apart from that, we also analyze the speed-up on the higher-dimensional relatives of the path and derive the first complete asymptotic characterisation of  $S_{\text{cov}}^{(k)}$ .

**Our Contributions.** In the following, we will describe our main results in more detail and sketch how to derive them. Our first result extends the example of Efremenko and Reingold in two parameters,  $n$  and  $k$ , and reveals an interesting dichotomy:

► **Theorem (Main Result 1).** *Consider a path with  $n$  vertices, where  $n \rightarrow \infty$ . Then the following results hold regardless of the loop-probability of the random walk:*

- For  $k = 2$ , the speed-up satisfies  $S_{\text{cov}}^{(k)} > 2$ .
- For  $k \geq 3$ , the speed-up satisfies  $S_{\text{cov}}^{(k)} < k$ .

The most challenging part of this result is the analysis for  $k = 2$ . We first prove that  $k$  discrete-time random walks explore the graph in nearly the same time as  $k$  continuous-time random walks (see Section 3, in particular Theorem 3.6, for a more precise statement). Then we relate the continuous-time random walks to a so-called multi-dimensional Gambler’s ruin problem, that was analyzed by Kmet and Petkovšek in [10]. This Gambler’s ruin problem provides us with a fairly accurate estimate of the cover time for the case where both random walks start from the same endpoint. However, to get a lower-bound on  $S_{\text{cov}}^{(k)}$ , we have to derive an upper bound on the cover time of  $k$  random walks for *any* start vertex. We eventually achieve this by a series of coupling arguments and reductions to the base case.

The sub-linear speed-up for  $k \geq 3$  is also derived via the relation to the Gambler’s ruin problem; one difficulty are cases where  $k$  is a very large constant. To bootstrap from smaller

## 44:4 Multiple Random Walks on Paths and Grids

$d = 2 / k \in$	$t_{\text{cov}}^{(k)}$	Speed-up	$d \geq 3 / k \in$	$t_{\text{cov}}^{(k)}$	Speed-up
$[1, \log^2 n]$	$\Theta\left(\frac{n \log^2 n}{k}\right)$	linear	$[1, n^{1-2/d} \log n]$	$\Theta\left(\frac{n \log n}{k}\right)$	linear
$[\log^2 n, n]$	$\Theta\left(\frac{n}{\log \frac{k}{\ln^2 n}}\right)$	logarithmic	$[n^{1-2/d} \log n, n]$	$\Theta\left(n^{2/d} / \log\left(\frac{k}{n^{1-2/d} \log n}\right)\right)$	logarithmic

■ **Figure 3** Overview of our results on  $t_{\text{cov}}^{(k)}$  for the two-dimensional grid/torus on the left hand side, and for the  $d$ -dimensional grid/torus,  $d \geq 3$ , on the right hand side.

to larger values of  $k$ , we derive a general result that the speed-up of the hitting time between the two endpoints is sub-linear in  $k$  (Lemma 4.5).

Finally, the fact that, as  $n \rightarrow \infty$ , the effect of the loop-probability becomes negligible, rests on the connection to continuous-time random walks; in fact, it holds not only for paths but also for arbitrary graphs (see Section 3.2 for our results on the impact of loop-probabilities).

For two-dimensional grids, we provide the first asymptotic characterization of  $S^{(k)}$  that holds for any  $1 \leq k \leq n$  (previous work [1, 8] had only determined  $S^{(k)}$  up to logarithmic factors.) We find that for  $1 \leq k \leq \log^2 n$ , the speed-up is linear, while for  $\log^2 n \leq k \leq n$ , the speed-up is logarithmic. While a weaker dichotomy was established before [8], our new result demonstrates a very sharp transition behavior between these two extreme cases.

► **Theorem (Main Result 2).** *Consider the two-dimensional grid/torus with  $n$  vertices as  $n \rightarrow \infty$ . Then:*

- For any  $k \in [1, \log^2 n]$ ,  $t_{\text{cov}}^{(k)} = \Theta\left(\frac{n \log^2 n}{k}\right)$  and thus  $S_{\text{cov}}^{(k)} = \Theta(k)$ .
- For any  $k \in [\log^2 n, n]$ ,  $t_{\text{cov}}^{(k)} = \Theta\left(\frac{n}{\log(k/\log^2 n)}\right)$  and thus  $S_{\text{cov}}^{(k)} = \Theta(\log^2 n \cdot \log(k/\log^2 n))$ .

The upper bounds on the cover time follow from a second-moment type argument about return times (a similar technique has been used by Cooper et al. [4]). Deriving the lower bounds appears to be more challenging, in particular for small values of  $k$ , where we elaborate on our connection to continuous-time random walks and apply a lower-bound technique from Zuckerman [20] to multiple random walks.

For three or higher dimensions, the situation is very similar; the only difference is that the transition point moves further up towards  $n$ , and that all cover times are reduced by a  $\log n$ -factor, which is in analogy to the behavior of single random walks.

► **Theorem (Main Result 3).** *Consider the  $d$ -dimensional grid/torus, with  $n$  vertices as  $n \rightarrow \infty$ , where  $d \geq 3$  is any fixed constant. Then:*

- For any  $k \in [1, n^{1-2/d} \log n]$ ,  $t_{\text{cov}}^{(k)} = \Theta\left(\frac{n \log n}{k}\right)$  and thus  $S_{\text{cov}}^{(k)} = \Theta(k)$ .
- For any  $k \in [n^{1-2/d} \log n, n]$ ,  $t_{\text{cov}}^{(k)} = \Theta\left(n^{2/d} / \log\left(\frac{k}{n^{1-2/d} \log n}\right)\right)$  and therefore  $S_{\text{cov}}^{(k)} = \Theta\left(n^{1-2/d} \cdot \log n \cdot \log\left(\frac{k}{n^{1-2/d} \log n}\right)\right)$ .

Figure 3 summarizes our results for two-dimensional grid/torus and  $d$ -dimensional grid/torus for  $d \geq 3$ .

**Organisation.** In Section 2 we provide the required mathematical notation and basic definitions. Section 3 deals with the connection between continuous and discrete-time multiple random walks. Then in Section 4 we analyze the two cases  $k = 2$  and  $k \geq 3$  for a path with  $n$  vertices. Section 5 contains our results for 2-dimensional grids (the results for  $d \geq 3$ -dimensional grids are derived in a similar fashion). We close our paper in Section 6 by pointing to some interesting open questions.

## 2 Notation and Definitions

Unless mentioned otherwise, we always consider simple random walks on an undirected, simple, connected and unweighted graph  $G = (V, E)$ , where  $n = |V|$  is the number of vertices. More specifically, a random walk starts from an arbitrary prespecified vertex and then at each step  $1, 2, \dots$  the random walk moves to a neighbor chosen uniformly at random. This can be encoded by a transition matrix  $P$ , where  $p_{u,v} = 1/\deg(u)$  if  $\{u, v\} \in E(G)$ ,  $p_{u,v} = 0$  otherwise. Further, we denote by  $p_{u,v}^t$  the probability for a random walk starting from  $u$  to be at vertex  $v$  at time-step  $t$ . Finally, we denote by  $\pi$  the *stationary distribution* that satisfies  $\pi(u) = \frac{\deg(u)}{2|E|}$ .

By  $\tau^{(k)}((u_1, \dots, u_k), v)$  we denote the first passage time, i.e., the first time one of  $k$  independent and concurrent random walks hits  $v$ , where the  $i$ -th random walk starts from vertex  $u_i$ . Then the *hitting time* is the expectation of the first passage time, in symbols

$$t_{\text{hit}}^{(k)}((u_1, \dots, u_k), v) := \mathbf{E} \left[ \tau^{(k)}((u_1, \dots, u_k), v) \right].$$

Similarly,  $\zeta^{(k)}(u_1, \dots, u_k)$  denotes the first time  $k$  independent and concurrent random walks have visited all vertices in  $V$ , where again the  $i$ -th random walk starts from vertex  $u_i$ . Then the *cover time* is the expectation of that random variable, in symbols

$$t_{\text{cov}}^{(k)}(u_1, \dots, u_k) := \mathbf{E} \left[ \zeta^{(k)}(u_1, \dots, u_k) \right].$$

For single random walks, we will often drop the <sup>(1)</sup>-superscript and simply write  $t_{\text{cov}}(u)$  and  $t_{\text{hit}}(u, v)$ . Finally, we define  $t_{\text{cov}} := \max_u t_{\text{cov}}(u)$  and  $t_{\text{cov}}^{(k)} := \max_u t_{\text{cov}}^{(k)}(u, \dots, u)$ . Since in many cases we will work with start configurations where all  $k$  random walks start from the same vertex, we may just write  $\vec{u}$  instead of  $(u, \dots, u)$ .

The *speed-up* for the cover time is the following function of  $k$  (and the underlying graph):

$$S_{\text{cov}}^{(k)} := \frac{t_{\text{cov}}}{t_{\text{cov}}^{(k)}} = \frac{\max_u t_{\text{cov}}(u)}{\max_u t_{\text{cov}}^{(k)}(\vec{u})}.$$

This has been the common definition of speed-up introduced by Alon et al. [1] and the focus of subsequent works [7, 8, 19]. However, we will also briefly discuss a slightly different version of  $S_{\text{cov}}^{(k)}$  at the end of Section 4. Similarly, the speed-up for the hitting time between  $u$  and  $v$  is the following function:

$$S_{\text{hit}}^{(k)}(u, v) := \frac{t_{\text{hit}}(u, v)}{t_{\text{hit}}^{(k)}(\vec{u}, v)}.$$

If the random walk is *lazy*, i.e. it stays with some non-zero probability  $\alpha > 0$  at the current vertex and otherwise moves to a randomly chosen neighbor, we adjust the notation and write  $t_{\text{hit}}^{(\alpha)}(u, v)$ ,  $t_{\text{hit}}^{(k, \alpha)}(\vec{u}, v)$  and  $S_{\text{hit}}^{(k, \alpha)}(u, v)$  to reflect the dependency on  $\alpha$ .

## 3 Continuous-Time Multiple Random Walks

### 3.1 Relating Continuous to Discrete-Time Multiple Random Walks

In the following we will relate the original,  $k$  discrete-time random walks to a continuous-time variant. In the continuous-time variant, the waiting time between any two transitions of a random walk are independent and identically distributed exponential random variables. We will denote the corresponding quantities by a superscript  $\widetilde{\phantom{x}}$ , e.g.,  $\widetilde{t_{\text{cov}}}^{(k)}$  is the cover time of

$k$  concurrent continuous-time random walks. In terms of all conventional and “reasonable” definitions of speed-ups, it turns out that the mean of the waiting time scales all hitting and cover times linearly, and is therefore irrelevant. Therefore the mean 1 is chosen for the sake of convenience unless mentioned otherwise. That means, the waiting time between any two transitions of a random walk are independent exponential random variables with parameter 1 (for further details we refer the reader to [13, Chapter 20.1]).

Furthermore, it is a well-known fact from the theory of Poisson processes that if we focus on one of the  $k$  walks, then the times at which a transition happens follows a Poisson point process. In fact something analogous holds if we consider all transitions among  $k$  walks, as described in the next lemma:

► **Lemma 3.1.** *Let  $G$  be any graph. Then continuous-time multiple random walks, with  $\lambda$  as the waiting-time parameter for each of the  $k$  walks, are a Poisson point process in which in each transition exactly one of the  $k$  walks performs a transition, and the waiting-times between two successive transitions are exponentially distributed random variables with parameter  $k\lambda$ .*

We continue to establish that for single random walks all hitting times and cover times are identical for discrete-time and continuous-time random walks.

► **Lemma 3.2.** *For a single random walk on an arbitrary graph  $G$ , we have the following equalities: (i) for any pair of vertices  $u, v \in V$ ,  $t_{\text{hit}}(u, v) = \widetilde{t}_{\text{hit}}(u, v)$  and (ii) for any node  $u \in V$ ,  $t_{\text{cov}}(u) = \widetilde{t}_{\text{cov}}(u)$ .*

The proof of Lemma 3.2 makes use of a natural coupling between two types of random walks and Wald’s equation ([13, Exercise 6.6]).

Before we connect the cover times and hitting times of continuous to their discrete-time counterparts, we recall the following useful lower bound on the cover time of  $k$  random walks (recall our notation  $\vec{u} = (u, \dots, u)$ ):

► **Lemma 3.3** ([8, Theorem 4.2]<sup>1</sup>). *Let  $G$  be any graph with  $n$  vertices and  $u$  be an arbitrary start vertex of  $k$  independent discrete-time random walks, where  $n^\varepsilon \leq k \leq n$  for some arbitrary  $\varepsilon > 0$ . Then*

$$\Pr \left[ t_{\text{cov}}^{(k)}(\vec{u}) \geq \frac{\varepsilon}{8} \cdot \frac{n}{k} \cdot \log n \right] \geq 1 - \exp \left( -n^{-\varepsilon/8} \right).$$

In the following lemma we show that the cover times of the continuous-time random walk are not much larger than those of a discrete-time random walk.

► **Lemma 3.4.** *For any graph  $G$  and  $1 \leq k \leq n$ ,*

$$\widetilde{t}_{\text{cov}}^{(k)}(\vec{u}) \leq 42 \cdot t_{\text{cov}}^{(k)}(\vec{u}).$$

Furthermore, if the graph  $G$  and  $k$  satisfy  $\Pr \left[ t_{\text{cov}}^{(k)}(\vec{u}) \geq f(n) \right] \geq 1 - o(1)$  for some function  $f(n) = \omega(\log n)$ , then the above inequality can be strengthened to:

$$\widetilde{t}_{\text{cov}}^{(k)}(\vec{u}) \leq (1 + o(1)) \cdot t_{\text{cov}}^{(k)}(\vec{u}).$$

---

<sup>1</sup> While the result stated in [8] involves the worst-case start vertices, the proof does not make use of this fact which is why we immediately obtain this slightly more general statement.



**Proof.** Fix any integer  $t \geq \frac{1}{8} \log n$ . For any of the  $k$  continuous-time random walks, let  $\mathcal{E}_t^i$  be the event that occurs if the time for the  $i$ -th random walk to perform the first  $t$  transition is at most  $t + 20t^{2/3}(\log n)^{1/3}$ . To this end, note that the number of transitions within time  $t + 20t^{2/3}(\log n)^{1/3}$  has a Poisson distribution with mean  $\mu = t + 20t^{2/3}(\log n)^{1/3}$ . Hence by a Chernoff bound for Poisson random variables ([2, Theorem A.1.15])

$$\begin{aligned} \Pr[\overline{\mathcal{E}}_t^i] &= \Pr[P < t] \leq \Pr\left[P < \left(1 - \frac{20t^{2/3}(\log n)^{1/3}}{\mu}\right) \cdot \mu\right] \\ &\leq \exp\left(-\frac{400t^{4/3}(\log n)^{2/3}}{2\mu}\right) \leq \exp\left(-4t^{1/3}(\log n)^{2/3}\right), \end{aligned}$$

where in the last inequality we used the fact that the assumption  $t \geq \frac{1}{8} \log n$  implies  $\mu \leq 41t$ . Let  $\mathcal{E}_t := \bigcup_{i=1}^k \mathcal{E}_t^i$ . By a Union bound over all  $k \leq n$  random walks,

$$\Pr[\mathcal{E}_t] \leq n \cdot \exp\left(-4t^{1/3}(\log n)^{2/3}\right) = n \cdot n^{-4 \cdot (t/\log n)^{1/3}} \leq n^{-1}.$$

To relate  $\widetilde{t}_{\text{cov}}^{(k)}(\vec{u})$  (cover time for  $k$  continuous-time walks) to  $t_{\text{cov}}^{(k)}(\vec{u})$  (cover time for  $k$  discrete-time walks), we consider again the straightforward coupling, where the random walk  $i$  (in discrete and continuous case) performs the same sequence of transitions. In other words, the two trajectories in discrete-time and continuous-time are identical, and only the times at which the transitions happen will be different. With this, we can now begin to upper bound  $\widetilde{t}_{\text{cov}}^{(k)}(\vec{u})$  by decomposing the expectation:

$$\begin{aligned} \widetilde{t}_{\text{cov}}^{(k)}(\vec{u}) &= \sum_{t=0}^{\infty} \Pr[\zeta^{(k)}(\vec{u}) = t] \cdot \mathbf{E}\left[\widetilde{t}_{\text{cov}}^{(k)}(\vec{u}) \mid \zeta^{(k)}(\vec{u}) = t\right] \\ &\leq \Pr\left[\zeta^{(k)}(\vec{u}) \leq \frac{1}{8} \cdot \log n\right] \cdot \mathbf{E}\left[\widetilde{t}_{\text{cov}}^{(k)}(\vec{u}) \mid \zeta^{(k)}(\vec{u}) \leq \frac{1}{8} \cdot \log n\right] \\ &\quad + \sum_{t=\frac{1}{8} \cdot \log n}^{\infty} \Pr[\zeta^{(k)}(\vec{u}) = t] \cdot \mathbf{E}\left[\widetilde{t}_{\text{cov}}^{(k)}(\vec{u}) \mid \zeta^{(k)}(\vec{u}) = t\right]. \end{aligned}$$

To bound  $\Pr[\zeta^{(k)}(\vec{u}) \leq \frac{1}{8} \cdot \log n]$ , we can apply the bound from Theorem 3.3 for  $k = n$  (since  $\zeta^{(k)}(\vec{u})$  is stochastically dominated by  $\zeta^{(n)}(\vec{u})$ ), and combine it with the simple bound  $\mathbf{E}\left[\widetilde{t}_{\text{cov}}^{(k)}(\vec{u}) \mid \zeta^{(k)}(\vec{u}) \leq \frac{1}{8} \cdot \log n\right] \leq \mathbf{E}\left[\widetilde{t}_{\text{cov}}^{(k)}(\vec{u})\right]$  to obtain

$$\widetilde{t}_{\text{cov}}^{(k)}(\vec{u}) \leq e^{-n^{1/8}} \mathbf{E}\left[\widetilde{t}_{\text{cov}}^{(k)}(\vec{u})\right] + \sum_{t=\frac{1}{8} \cdot \log n}^{\infty} \Pr[\zeta^{(k)}(\vec{u}) = t] \mathbf{E}\left[\widetilde{t}_{\text{cov}}^{(k)}(\vec{u}) \mid \zeta^{(k)}(\vec{u}) = t\right].$$

In the following, we shall derive an upper bound on  $\mathbf{E}\left[\widetilde{t}_{\text{cov}}^{(k)}(\vec{u}) \mid \zeta^{(k)}(\vec{u}) = t\right]$ . First, by conditioning on the event  $\mathcal{E}_t$  we obtain

$$\begin{aligned} &\mathbf{E}\left[\widetilde{t}_{\text{cov}}^{(k)}(\vec{u}) \mid \zeta^{(k)}(\vec{u}) = t\right] \\ &= \mathbf{E}\left[\widetilde{t}_{\text{cov}}^{(k)}(\vec{u}) \mid \zeta^{(k)}(\vec{u}) = t, \mathcal{E}_t\right] \Pr[\mathcal{E}_t] + \mathbf{E}\left[\widetilde{t}_{\text{cov}}^{(k)}(\vec{u}) \mid \zeta^{(k)}(\vec{u}) = t, \overline{\mathcal{E}}_t\right] \Pr[\overline{\mathcal{E}}_t] \quad (1) \end{aligned}$$

We shall now proceed by bounding the two conditional expectations in eq. (1). First, we have

$$\mathbf{E}\left[\widetilde{t}_{\text{cov}}^{(k)}(\vec{u}) \mid \zeta^{(k)}(\vec{u}) = t, \mathcal{E}_t\right] \leq (t + 20t^{2/3}(\log n)^{1/3}),$$

since conditional on  $\mathcal{E}_t$ , all  $k$  random walks perform at least  $t$  transitions within the time-interval  $[0, t + 20t^{2/3}(\log n)^{1/3}]$ .

To upper bound  $\mathbf{E} \left[ \widetilde{t_{\text{cov}}^{(k)}}(\vec{u}) \mid \zeta^{(k)}(u) = t, \overline{\mathcal{E}_t} \right]$  note that  $\overline{\mathcal{E}_t}$  only affects the random walks within the time interval  $[0, t + 20t^{2/3}(\log n)^{1/3}]$ . Therefore we have,

$$\mathbf{E} \left[ \widetilde{t_{\text{cov}}^{(k)}}(\vec{u}) \mid \zeta^{(k)}(u) = t, \overline{\mathcal{E}_t} \right] \leq \left( t + 20t^{2/3}(\log n)^{1/3} + \mathbf{E} \left[ \widetilde{t_{\text{cov}}^{(k)}}(\vec{u}) \mid \zeta^{(k)}(u) = t \right] \right).$$

Combining these two bounds and plugging these into equation (1) yields

$$\begin{aligned} \mathbf{E} \left[ \widetilde{t_{\text{cov}}^{(k)}}(\vec{u}) \mid \zeta^{(k)}(u) = t \right] &\leq (t + 20t^{2/3}(\log n)^{1/3}) \cdot (1 - \mathbf{Pr}[\overline{\mathcal{E}_t}]) + \\ &\quad \left( t + 20t^{2/3}(\log n)^{1/3} + \mathbf{E} \left[ \widetilde{t_{\text{cov}}^{(k)}}(\vec{u}) \mid \zeta^{(k)}(\vec{u}) = t \right] \right) \cdot \mathbf{Pr}[\overline{\mathcal{E}_t}] \\ &= t + 20t^{2/3}(\log n)^{1/3} + \mathbf{E} \left[ \widetilde{t_{\text{cov}}^{(k)}}(\vec{u}) \mid \zeta^{(k)}(\vec{u}) = t \right] \cdot \mathbf{Pr}[\overline{\mathcal{E}_t}]. \end{aligned}$$

Using  $\mathbf{Pr}[\overline{\mathcal{E}_t}] \leq n^{-1}$  and rearranging gives

$$\mathbf{E} \left[ \widetilde{t_{\text{cov}}^{(k)}}(\vec{u}) \mid \zeta^{(k)}(\vec{u}) = t \right] \leq \frac{1}{1 - n^{-1}} \cdot \left( t + 20t^{2/3}(\log n)^{1/3} \right).$$

Using this bound and returning to the upper bound on  $\widetilde{t_{\text{cov}}^{(k)}}(\vec{u})$  yields

$$\begin{aligned} \widetilde{t_{\text{cov}}^{(k)}}(\vec{u}) \cdot (1 - e^{-n^{1/8}}) &\leq \sum_{t=\frac{1}{8} \cdot \log n}^{\infty} \mathbf{Pr}[\zeta^{(k)}(\vec{u}) = t] \cdot \frac{1}{1 - n^{-1}} \cdot \left( t + 20t^{2/3}(\log n)^{1/3} \right) \quad (2) \\ &\leq \sum_{t=1}^{\infty} \mathbf{Pr}[\zeta^{(k)}(\vec{u}) = t] \cdot \frac{41t}{1 - n^{-1}}, \end{aligned}$$

where the final step holds because  $t + 20t^{2/3}(\log n)^{1/3} \leq 41t$ . Hence,  $\widetilde{t_{\text{cov}}^{(k)}}(\vec{u}) \leq 42 \cdot t_{\text{cov}}^{(k)}(\vec{u})$ .

To prove the tighter inequality, we proceed similarly. Details can be found in the full version.  $\blacktriangleleft$

The next lemma is the corresponding lower bound variant of Lemma 3.4:

► **Lemma 3.5.** *For any graph  $G$  and  $1 \leq k \leq n$ , there is a constant  $c > 0$  (independent of  $k$  and  $n$ ) so that*

$$\widetilde{t_{\text{cov}}^{(k)}}(\vec{u}) \geq c \cdot t_{\text{cov}}^{(k)}(\vec{u}).$$

Furthermore, if the graph  $G$  and  $k$  satisfy  $\mathbf{Pr} \left[ t_{\text{cov}}^{(k)}(\vec{u}) \geq f(n) \right] \geq 1 - o(1)$  for some function  $f(n) = \omega(\log n)$ , then the above inequality can be strengthened to:

$$\widetilde{t_{\text{cov}}^{(k)}}(\vec{u}) \geq (1 - o(1)) \cdot t_{\text{cov}}^{(k)}(\vec{u}).$$

The proof of this Lemma is similar to that of Lemma 3.4, and is therefore given in the appendix only. It might be worth mentioning that it is not possible to drop the upper bound on  $k$  in the precondition of Lemma 3.5; clearly, as  $k \rightarrow \infty$ ,  $t_{\text{cov}}^{(k)}(u) \geq \text{diam}$ , where  $\text{diam}$  is the diameter of the underlying graph, however, as  $k \rightarrow \infty$ ,  $\widetilde{t_{\text{cov}}^{(k)}}(u) \rightarrow 0$ .

Combining the three previous lemmas above, we conclude the following.

► **Theorem 3.6.** *Let  $G$  be any graph. For any speed-up of the cover time with  $1 \leq k \leq o(n)$ ,  $S_{\text{cov}}^{(k)} = (1 \pm o(1)) \cdot \widetilde{S_{\text{cov}}^{(k)}}$ . Moreover, for any  $1 \leq k \leq n$ ,  $S_{\text{cov}}^{(k)} = \Theta(\widetilde{S_{\text{cov}}^{(k)}})$ .*

Analogous results also hold for hitting times, but in this case one would need a lower bound like  $\mathbf{Pr} \left[ \tau(u, v) \geq \frac{1}{8} \log n \right] \geq 1 - o(1)$ , which may not be possible for all pairs of vertices.

### 3.2 The Effect of Loop Probabilities

Our first result demonstrates that the influence of the loop probabilities on the speed-up diminishes, as  $n \rightarrow \infty$ .

► **Theorem 3.7.** *Let  $G$  be any graph and  $0 \leq \alpha_1 < \alpha_2 \leq 1$  be any pair of loop probabilities. Then for any pair of vertices  $u, v$ ,  $S_{\text{cov}}^{(k, \alpha_1)}(u) = \Theta(S_{\text{cov}}^{(k, \alpha_2)}(u))$ . Further, if  $\Pr \left[ t_{\text{cov}}^{(k, 0)}(\vec{u}) \geq f(n) \right] \geq 1 - o(1)$  for some  $f(n) = \omega(\log n)$ , then  $S_{\text{cov}}^{(k, \alpha_1)}(u) = (1 + o(1)) \cdot S_{\text{cov}}^{(k, \alpha_2)}(u)$ .*

The proof of Theorem 3.7 is based on the following two steps. Firstly, we use the connection between continuous-time and (non-lazy) discrete-time random walks derived above. Secondly, we make use of the fact that in continuous-time, increasing the loop-probabilities from 0 to some value is just a scaling operation, which leaves the speed-up unchanged.

For hitting times, we can prove an even stronger result stating that the speed-up of hitting times between any pair of vertices is non-decreasing in the loop-probability.

► **Lemma 3.8.** *Let  $G$  be any graph. Then for any combination of  $n$  and  $k$ , the speed-up  $S_{\text{hit}}^{(k, \alpha)}(u, v)$  of any hitting time pair for random walks with loop probability  $\alpha > 0$  is at least as large as the speed-up  $S_{\text{hit}}^{(k, 0)}(u, v)$  in the non-lazy setting.*

Similar to the proof of Lemma 3.2, the proof of Lemma 3.8 involves a natural coupling between lazy and non-lazy random walks and an application of Wald’s equation.

We now establish a useful duality between ‘very lazy’ random walks and continuous-time random walks.

► **Lemma 3.9.** *Let  $G$  be any graph and  $u, v$  be any two vertices. Then  $\lim_{\alpha \rightarrow 1} S_{\text{hit}}^{(k, \alpha)}(u, v) = \widetilde{S}_{\text{hit}}^{(k)}(u, v)$  and  $\lim_{\alpha \rightarrow 1} S_{\text{cov}}^{(k, \alpha)} = \widetilde{S}_{\text{cov}}^{(k)}$ .*

The proof of this lemma revolves around the idea that if the loop-probability is sufficiently large, then with high probability in each step among the  $k$  discrete-time multiple random walks at most one will move to a neighboring vertex. Hence there are no “concurrency effects”, and the discrete-time and continuous-time random walks behave correspondingly.

## 4 Speed-up on the Path

For convenience, we shall consider a path with  $n + 1$  nodes labelled from 0 to  $n$ , and  $n$  is even. We defer the straightforward but tedious adjustments for odd  $n$  to the full version.

Before delving into further specifics of the analyses for  $k = 2$  and  $k = 3$ , let us briefly mention the following fact for the single random walks ( $k = 1$ ). To determine the cover time from the vertex, say,  $n/2$ , we can use the Gambler’s ruin problem to infer that the time until one of the vertices 0 or  $n$  is reached equals  $(n/2 - 0) \cdot (n - n/2) = n^2/4$ . Then, the time until the opposite endpoint is reached equals  $t_{\text{hit}}(0, n) = t_{\text{hit}}(n, 0) = n^2$  (cf. [15]). Combining this, we arrive at the following simple yet crucial observation:

► **Observation 4.1.** *For a single random walk ( $k = 1$ ) on a path with endpoints 0 and  $n$ ,  $t_{\text{cov}}(n/2) = \frac{5}{4} \cdot n^2$ . Furthermore,  $t_{\text{cov}}(u)$  is maximized for  $u = n/2$ .*

Recalling the definition of  $S_{\text{cov}}^{(k)}$ , the strategy to establish a sub-linear speed-up for  $k \geq 3$  is to find a vertex  $u$  so that  $t_{\text{cov}}^{(k)}(\vec{u}) > \frac{1}{k} \cdot \frac{5}{4} \cdot n^2$ . A both natural and convenient candidate is to pick the endpoint, and we shall indeed establish in the next subsection that  $t_{\text{cov}}^{(k)}(\vec{0}) = t_{\text{hit}}^{(k)}(\vec{0}, n) > \frac{1}{k} \cdot \frac{5}{4} \cdot n^2$ .

$k$	1	2	3	4	5	6	7	8	9
$c_k$	1	1.178	1.349	1.512	1.670	1.823	1.973	2.118	2.261

■ **Figure 4** The numerical values for  $c_k$  as given in [10] rounded down to the third significant digit.

In contrast to this, establishing a super-linear speed-up for  $k = 2$  turns out to be more demanding because in this case we have to upper bound  $t_{\text{cov}}^{(k)}(\vec{u})$  for *any* vertex  $u$ .

### 4.1 $k \geq 3$ : Sub-linear Speed-up

In this section we shall prove that the speed-up for cover times on paths is sub-linear as soon as  $k \geq 3$ . We should mention that for sufficiently large values of  $k$ , i.e.,  $k$  being super-constant, [1] proved that on the cycle with  $n$  vertices that  $S^k = \Theta(\log k)$ , as  $n \rightarrow \infty$ . Unfortunately, it seems difficult to apply the result (or proof) of [1] to the case of a path with a small constant  $k$ . Here we will show the following theorem, that captures all values of  $k \geq 3$ :

► **Theorem 4.2.** *Consider a discrete-time random walk on a path with endpoints 0 and  $n$ . Then for any  $k \geq 3$ , as  $n \rightarrow \infty$ ,  $t_{\text{cov}}^{(k)}(\vec{0}) = t_{\text{hit}}^{(k)}(\vec{0}, n) > \frac{5/4}{k} \cdot n^2 - o(n^2)$ .*

Since  $t_{\text{cov}}(n/2) = \frac{5}{4} \cdot n^2$  by Observation 4.1, the theorem above immediately implies the following result:

► **Theorem 4.3.** *For any path with endpoints 0 and  $n$  and  $k \geq 3$ , as  $n \rightarrow \infty$ ,  $S_{\text{cov}}^{(k)} < k$ .*

Before proving Theorem 4.2, we list three auxiliary results. The theorem below follows from a work by Kmet and Petkovsek [10] on the  $k$ -dimensional Gambler's ruin problem.

► **Theorem 4.4** (cf. [10]). *Consider a continuous-time random walk on a path with endpoints 0,  $n$ . Then for any  $1 \leq k \leq 10$ ,*

$$\widetilde{t}_{\text{cov}}^{(k)}(\vec{0}) = \widetilde{t}_{\text{hit}}^{(k)}(\vec{0}, n) > \frac{c_k}{k} \cdot n^2 - o(n^2/k),$$

where the  $c_k$ 's are the numerical values given in the table below.

The following lemma establishes that the speed-up is at most sub-linear, however, this is for the hitting time between two endpoints.

► **Lemma 4.5.** *For the path with endpoints 0 and  $n$ , we have for any pair of integers  $\ell \geq 1$  and  $k \geq 1$ ,  $t_{\text{hit}}^{(\ell \cdot k)}(\vec{0}, n) \geq \frac{1}{\ell} \cdot t_{\text{hit}}^{(k)}(\vec{0}, n)$ .*

**Proof.** Let  $X = \tau^{(k)}(\vec{0}, n)$  be the random variable describing the first visit to  $n$  of a random walk starting from 1; so  $\mathbf{E}[X] = t_{\text{hit}}^{(k)}(\vec{0}, n)$ . For the  $\ell \cdot k$  random walks, let  $X_1, X_2, \dots, X_{\ell \cdot k}$  be  $\ell \cdot k$  independent copies of  $X$ . We have:

$$\begin{aligned} t_{\text{hit}}^{(\ell \cdot k)}(\vec{0}, n) &= \sum_{i=1}^{\infty} \Pr[\min\{X_1, \dots, X_k\} = i] + \sum_{i=1}^{\infty} \Pr[\min\{X_1, \dots, X_k\} > i] \\ &= 1 + \sum_{i=1}^{\infty} \Pr[X > i]^\ell \end{aligned}$$

Note that  $\Pr[X > \ell \cdot i]$  can be upper bounded by  $\Pr[X > i]^\ell$ . This follows by dividing the time into consecutive blocks of length  $i$  each, and noting that the probability that a fixed random walk does not hit  $n$  in any of the blocks is upper bounded by  $\Pr[X > i]$ . Therefore,

$$\begin{aligned} t_{\text{hit}}^{(\ell,k)}(\vec{0}, n) &\geq 1 + \sum_{i=1}^{\infty} \Pr[X > \ell \cdot i] = 1 - \sum_{i=1}^{\infty} \Pr[X = \ell \cdot i] + \sum_{i=1}^{\infty} \Pr[X \geq \ell \cdot i] \\ &\geq \sum_{i=1}^{\infty} \Pr[X \geq \ell \cdot i] = \mathbf{E}[X/\ell] = \frac{1}{\ell} \cdot t_{\text{hit}}^{(k)}(\vec{0}, n). \quad \blacktriangleleft \end{aligned}$$

Finally, we shall need the following simple lower bound on  $t_{\text{hit}}^{(k)}(\vec{0}, n)$ , that will enable us to apply Lemma 3.4:

► **Lemma 4.6.** *For any  $1 \leq k \leq n$ ,  $\Pr\left[t_{\text{hit}}^{(k)}(\vec{0}, n) \geq n^2/\log^2 n\right] \geq 1 - o(1)$ .*

The basic outline of the proof of Theorem 4.2 is now as follows. We use the lower bound from Theorem 4.4 for smaller values of  $k$  as a base case. Then we use Lemma 3.4 to derive corresponding lower bounds for discrete-time random walks. Finally, we use Lemma 4.5 to “bootstrap” the lower bounds from smaller values of  $k$  to larger values.

## 4.2 $k = 2$ : Super-linear Speed-up

First let us recall that  $c_2$  is the constant from the two-dimensional Gambler’s ruin problem and was shown to be at most 1.179 [10] (see also Theorem 4.4 and Figure 4 in this manuscript).

We first derive several upper bounds on the cover time for four special cases. The proofs of these three estimates are non-trivial but similar. In essence, they use couplings in order to reduce the cover time to the “base case” where the two random walks start from the same endpoint, for which we have a fairly accurate estimate (Theorem 4.4).

► **Lemma 4.7.** *Let  $G$  be a path with endpoints 0 and  $n$ . Then, as  $n \rightarrow \infty$ :*

1.  $t_{\text{cov}}^{(2)}(n/2, n/2) \leq \left(\frac{3}{8} + \frac{3c_2}{16}\right) n^2 + o(n^2) \leq 0.59602 \cdot n^2 + o(n^2)$ .
2.  $t_{\text{cov}}^{(2)}(n/4, n/4) \leq 0.6128 \cdot n^2 + o(n^2)$ .
3. For any  $n/4 \leq w \leq n/2$ ,

$$t_{\text{cov}}^{(2)}(w, n/2) \leq \frac{1}{2} \cdot \left( \left(2 - \frac{c_2}{2}\right) \cdot \frac{n^2}{4} \right) + \frac{1}{2} \left( \frac{1}{2} \cdot w \cdot (n - w) + \frac{1}{2} \cdot \frac{n}{2} \cdot \frac{n}{2} + \frac{c_2}{2} \cdot n^2 \right) + o(n^2),$$

where the cover time is under the implicit assumption that all vertices between  $w$  and  $n/2$  are already visited.

4. For any  $0 \leq w < n/4$ ,  $t_{\text{cov}}^{(2)}(0, w) \leq \frac{c_2}{2} \cdot n^2 - \frac{c_2}{4} \cdot w^2 + o(n^2)$ , where the cover time is under the implicit assumption that all vertices between 0 and  $w$  are already visited.

Equipped with the three auxiliary estimates above, we now analyze  $t_{\text{cov}}^{(2)}(w, w)$ .

► **Proposition 4.8.** *Let  $G$  be a path with endpoints 0 and  $n$ . Then, as  $n \rightarrow \infty$ :*

1. For any  $n/4 \leq w \leq n/2$ ,  $t_{\text{cov}}^{(2)}(w, w) \leq 0.6244 \cdot n^2 + o(n^2) < \frac{5}{8} \cdot n^2$ .
2. For any  $0 \leq w < n/4$ ,  $t_{\text{cov}}^{(2)}(w, w) \leq 0.6128 \cdot n^2 + o(n^2) < \frac{5}{8} \cdot n^2$ .

With the estimates from Lemma 4.7 at hand, the proof of the first statement is essentially a repeated application of the corresponding bounds in Lemma 4.7, depending which of the two endpoints 0 or  $n/4$  the first and second random walk reach first (cf. appendix).

Finally, by using the two results in the above propositions and noting the simple symmetry that  $t_{\text{cov}}^{(2)}(w, w) = t_{\text{cov}}^{(2)}(n - w, n - w)$  for any  $0 \leq w \leq n/2$ , we obtain:

► **Theorem 4.9.** *For the path with vertices  $0, \dots, n$ , as  $n \rightarrow \infty$ , then for any  $1 \leq w \leq n$ ,  $t_{\text{cov}}^{(2)}(w, w) < \frac{5}{8} \cdot n^2$ . Consequently, as  $n \rightarrow \infty$ ,  $S_{\text{cov}}^{(k)} > 2$ .*

Let us briefly mention that even if we were to redefine the speed-up as the quantity  $S_{\text{cov}}^{(k)} := \max_{u \in V} \frac{t_{\text{cov}}(u)}{t_{\text{cov}}^{(k)}(\bar{u})}$ , our results above for  $u = n/2$  would imply a super-linear speed-up.

## 5 Speed-up on d-Dimensional Grid/Torus

A  $d$ -dimensional grid is the graph with vertex set  $V = [-n^{1/d}/2, +n^{1/d}/2]^d$ , and two vertices are connected iff they differ in one coordinate by 1 and all other coordinates are identical. A  $d$ -dimensional torus is almost same as the  $d$ -dimensional grid, but additionally we have “wrap-around” edges, e.g., for  $d = 2$ , we have additional edges between  $(x, n/2)$  and  $(x, -n/2)$ , and  $(n/2, x)$  and  $(-n/2, x)$  for any  $-n/2 \leq x \leq n/2$ .

Due to space limitations, we only sketch the analysis for  $d = 2$  here. The analysis for  $d \geq 3$  is quite similar and even slightly easier than the one for  $d = 2$ . To avoid any periodicity issues, we shall always consider lazy random walks with loop probability  $1/2$ .

### 5.1 Preliminaries

► **Lemma 5.1.** *For any graph  $G$  and any integer  $t$ ,  $\Pr[\tau(u, v) \leq t] \geq \frac{\sum_{s=0}^t p_{uv}^s}{1 + \mathbf{E}[R_u(t)]}$ , where  $\mathbf{E}[R_u(t)] = \sum_{i=1}^t p_{u,u}^i$  is the expected number of returns to  $u$  until step  $t$ .*

We next note an elementary fact about the limiting behaviour of the  $t$ -step transition probabilities. It can be easily derived by relating the random walk to the infinite grid  $\mathbb{Z}^d$  and then applying the central limit theorem (e.g. [12]).

► **Lemma 5.2.** *Consider a  $d$ -dimensional grid or torus, where  $d$  is constant, and let  $n \rightarrow \infty$  be sufficiently large. Then for any pair of vertices  $u, v \in V$ , there is a constant  $c_1 > 0$  so that  $p_{u,v}^t \leq \pi(v) + c_1 \cdot t^{-d/2}$ . Moreover, for any pair of vertices  $u, v \in V$ , there is a constant  $c > 0$  so that*

$$p_{u,v}^t \geq c \cdot t^{-d/2} \cdot \exp\left(-\frac{d \cdot \|u - v\|_2^2}{t}\right) - c \cdot t^{-(d+2)/2}.$$

### 5.2 $d = 2$ and $k \in [1, \log^2 n]$ : Linear Speed-Up

We begin this part by deriving the upper bound on the cover time. First, we point out that for  $1 \leq k \leq \log n$ , the desired bound  $t_{\text{cov}}^{(k)} = O(n \log^2 n/k)$  follows from [1, Theorem 4]. To cover also the missing regime  $\log n \leq k \leq \log^2 n$ , we show the following lemma:

► **Lemma 5.3.** *For  $k \in [1, \log^2 n]$  the cover time of  $k$  independent lazy random walks on a 2-dimensional torus is  $O(n \log^2 n/k)$ .*

This result is a fairly straightforward application of Lemma 5.1. The derivation of the lower bound is more involved, and is based on the following result:

► **Lemma 5.4** ([20, Lemma 2]). *Let  $V' \subseteq V$  such that  $|V'| \geq n^\delta, \delta > 0$  and let  $t$  such that for  $i \in V'$ , at most  $1/n^\beta$  fraction of the  $j \in V'$  satisfy  $t_{\text{hit}}(u, v) < t$ . Then for any start vertex  $v \in V$ ,  $t_{\text{cov}}(v) \geq t \cdot (\gamma \ln n - 2)$ , where  $\gamma = \min(\delta, \beta)$ .*

### 5.3 $d = 2$ and $k \in [\log^2 n, n]$ : Logarithmic Speed-Up

► **Lemma 5.5.** For  $k \in [\log^2 n, n]$  the cover time of  $k$  independent lazy random walks on a 2-dimensional torus is  $\Theta(n/\log(k/\log^2 n))$ .

## 6 Summary and Open Problems

While a focus of this work has been the speed-up of multiple random walks on paths and grids, an important general insight is the benefit of using continuous-time random walks. In that regard, Lemma 3.4, Lemma 3.5 and Theorem 3.6 (all holding for *arbitrary* graphs) might be useful for future work, since one can conveniently switch between continuous-time and discrete-time walks.

One interesting open problem is to analyze the speed-up on the cycle, where for  $k = 2$ , simulations indicate that the speed-up may be very close to 2. Secondly, in light of our result that already for the cover time on the path the speed-up can be super-linear, a logical next step would be to explore whether  $S_{\text{hit}}^{(k)}(u, v) \leq k$  holds (for the special case of the path and  $u$  and  $v$  being the endpoints, Lemma 4.5 provides a positive answer).

---

### References

- 1 Noga Alon, Chen Avin, Michal Koucký, Gady Kozma, Zvi Lotker, and Mark R. Tuttle. Many random walks are faster than one. *Combinatorics, Probability & Computing*, 20(4):481–502, 2011.
- 2 Noga Alon and Joel Spencer. *The Probabilistic Method*. John Wiley & Sons, 2nd edition, 2000.
- 3 Andrei Z. Broder, Anna R. Karlin, Prabhakar Raghavan, and Eli Upfal. Trading space for time in undirected s-t connectivity. *SIAM J. Comput.*, 23(2):324–334, 1994.
- 4 Colin Cooper, Alan M. Frieze, and Tomasz Radzik. Multiple random walks and interacting particle systems. In *Proc. 36th Intl. Colloquium on Automata, Languages and Programming (ICALP'09)*, pages 399–410, 2009.
- 5 Colin Cooper, David Ilcinkas, Ralf Klasing, and Adrian Kosowski. Derandomizing random walks in undirected graphs using locally fair exploration strategies. *Distributed Computing*, 24(2):91–99, 2011.
- 6 Artur Czumaj and Christian Sohler. Testing expansion in bounded-degree graphs. *Combinatorics, Probability & Computing*, 19(5-6):693–709, 2010.
- 7 Klim Efremenko and Omer Reingold. How well do random walks parallelize? In *13th International Workshop on on Randomization and Computation (RANDOM'09)*, pages 476–489, 2009.
- 8 Robert Elsässer and Thomas Sauerwald. Tight bounds for the cover time of multiple random walks. *Theor. Comput. Sci.*, 412(24):2623–2641, 2011.
- 9 Amos Israeli and Marc Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing (PODC'90)*, pages 119–131, 1990.
- 10 Andrej Kmet and Marko Petkovšek. Gambler's ruin problem in several dimensions. *Advances in Applied Mathematics*, 28:107–118, 2002.
- 11 Adrian Kosowski. *Time and Space-Efficient Algorithms for Mobile Agents in an Anonymous Network*. Research habilitation, Université Bordeaux 1, September 2013.
- 12 Gregory F. Lawler and Vlada Limic. *Random Walk: A Modern Introduction*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2010.

- 13 David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov Chains and Mixing Times*. AMS, 2008.
- 14 Pascal Lezaud. Chernoff-type bound for finite markov chains. *Ann. Appl. Probab.*, 8(3):849–867, 1989.
- 15 László Lovász. Random walks on graphs: A survey. *Combinatorics, Paul Erdős is Eighty*, 2:1–46, 1993.
- 16 George B. Mertzios, Sotiris E. Nikolettseas, Christoforos Raptopoulos, and Paul G. Spirakis. Determining majority in networks with local interactions and very small local memory. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP'14)*, pages 871–882, 2014.
- 17 Atish Das Sarma, Anisur Rahaman Molla, and Gopal Pandurangan. Fast distributed computation in dynamic networks via random walks. In *Proceedings of the 26th International Symposium on Distributed Computing (DISC'12)*, pages 136–150, 2012.
- 18 Atish Das Sarma, Danupon Nanongkai, Gopal Pandurangan, and Prasad Tetali. Distributed random walks. *J. ACM*, 60(1):2, 2013.
- 19 Thomas Sauerwald. Expansion and the cover time of parallel random walks. In *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing (PODC'10)*, pages 315–324, 2010.
- 20 David Zuckerman. A technique for lower bounding the cover time. *SIAM J. Discrete Math.*, 5(1):81–87, 1992.



# On the Size of Lempel-Ziv and Lyndon Factorizations\*

Juha Kärkkäinen<sup>1</sup>, Dominik Kempa<sup>2</sup>, Yuto Nakashima<sup>3</sup>,  
Simon J. Puglisi<sup>4</sup>, and Arseny M. Shur<sup>5</sup>

- 1 Helsinki Institute for Information Technology (HIIT), Helsinki, Finland; and Department of Computer Science, University of Helsinki, Helsinki, Finland  
juha.karkkainen@cs.helsinki.fi
- 2 Helsinki Institute for Information Technology (HIIT), Helsinki, Finland; and Department of Computer Science, University of Helsinki, Helsinki, Finland  
dominik.kempa@cs.helsinki.fi
- 3 Department of Informatics, Kyushu University, Fukuoka, Japan; and Japan Society for the Promotion of Science, Japan  
yuto.nakashima@inf.kyushu-u.ac.jp
- 4 Helsinki Institute for Information Technology (HIIT), Helsinki, Finland; and Department of Computer Science, University of Helsinki, Helsinki, Finland  
simon.puglisi@cs.helsinki.fi
- 5 Dept. of Algebra and Discrete Mathematics, Ural Federal University, Ekaterinburg, Russia  
arseny.shur@urfu.ru

---

## Abstract

Lyndon factorization and Lempel-Ziv (LZ) factorization are both important tools for analysing the structure and complexity of strings, but their combinatorial structure is very different. In this paper, we establish the first direct connection between the two by showing that while the Lyndon factorization can be bigger than the non-overlapping LZ factorization (which we demonstrate by describing a new, non-trivial family of strings) it is always less than twice the size.

**1998 ACM Subject Classification** F.2.2 [Nonnumerical Algorithms and Problems] Pattern Matching, G.2.1 [Combinatorics] Combinatorial Algorithms

**Keywords and phrases** Lempel-Ziv factorization, Lempel-Ziv parsing, LZ, Lyndon word, Lyndon factorization, Standard factorization

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.45

## 1 Introduction

Given a string (or word)  $x$ , a *factorization* of  $x$  partitions  $x$  into substrings  $f_1, f_2, \dots, f_t$ , such that  $x = f_1 f_2 \dots f_t$ . In the past 50 years or so, dozens of string factorizations have been studied, some purely out of combinatorial interest (e.g. [12, 1, 22, 2]) and others because the internal structure that they reveal allows the design of efficient string processing algorithms. Perhaps the two most important factorizations in string processing are the Lempel-Ziv (LZ) factorization [26] and the Lyndon factorization<sup>1</sup> [10].

---

\* This research was partially supported by the Academy of Finland through grant 294143 and by the RFBR grant 16-01-00795.

<sup>1</sup> Also known as the Standard factorization.



The LZ factorization has its origins in data compression, and is still used in popular file compressors<sup>2</sup> and as part of larger software systems (see, e.g., [7, 16] and references therein). More recently it has been used in the design of compressed data structures for indexed pattern matching [13] and other problems [5]. Each factor  $f_i$  in the LZ factorization must be as long as possible and must be either the first occurrence of a letter in  $x$  or occur in  $f_1 \dots f_{i-1}$ .<sup>3</sup> The Lyndon factorization, on the other hand, was first studied in the context of combinatorics on words [20, Sect. 5], and later found use in algorithms; for example, in a bijective variant of the Burrows-Wheeler transform [14, 18], in suffix sorting [21] and in repetition detection [4]. Each factor  $f_i$  in the Lyndon factorization must be a Lyndon word: a string that is lexicographically smaller than all its proper suffixes; and the factors must be lexicographically non-increasing. Lyndon words themselves have deep combinatorial properties [20] and have wide application [6, 9, 11, 15, 18, 19, 20, 23].

For some problems each factorization (Lempel-Ziv or Lyndon) leads to quite different solutions. Perhaps the best known example of this is the computation of all the maximal repetitions – also known as the “runs” – in a string. In 1999 Kolpakov and Kucherov proved that  $\rho(n)$ , the number of runs in a string of length  $n$ , is  $O(n)$ , and showed how to exploit the structure of the LZ factorization to compute all the runs in linear time [17]. Much more recently, Bannai et al. [3] used properties of the Lyndon factorization to obtain a much simpler constructive proof that  $\rho(n) < n$ . This later result also leads to a straight-forward linear-time algorithm for computing the runs from the Lyndon factorization [4].

Our overarching motivation in this paper is to obtain a deeper understanding of how these two fundamental factorizations – Lempel-Ziv and Lyndon – relate. Toward this aim, we ask: *by how much can the sizes of the factorizations of the same word differ?* Here the size of the Lempel-Ziv factorization  $s = p_1 \dots p_z$  is  $z$  and the size of the Lyndon factorization  $s = f_1^{e_1} \dots f_m^{e_m}$ , where each  $e_i$  is positive and each  $f_i$  is lexicographically strictly greater than  $f_{i+1}$ , is  $m$ . For most strings, the number of Lyndon factors is much smaller. Indeed, any string has a rotation with a Lyndon factorization of size one. So the actual question is how big can  $m$  be with respect to  $z$ . For a lower bound, we show that there are strings with  $m = z + \Theta(\sqrt{z})$ . Our main result is the upper bound: the inequality  $m < 2z$  holds for all strings. This result improves significantly a previous, indirect bound by I et al. [25], who showed that the number of Lyndon factors cannot be more than the size of the smallest straight line program (SLP). Since the smallest SLP is at most a logarithmic factor bigger than the LZ factorization [24, 8], this establishes an indirect, logarithmic factor bound, which we improve to a constant factor two.

## 2 Basic Notions

We consider finite strings over an alphabet  $\Sigma = \{a_1, \dots, a_n\}$ , which is linearly ordered:  $a_1 \prec a_2 \prec \dots \prec a_n$ . For strings, we use the array notation:  $s = s[1..|s|]$ , where  $|s|$  stands for the length of  $s$ . The empty string  $\varepsilon$  has length 0. Any pair  $i, j$  such that  $1 \leq i \leq j \leq |s|$  specifies a *substring*  $s[i..j]$  in  $s$ . A string  $u$  equal to some  $s[i..j]$  is a *factor* of  $s$  (a *prefix*, if  $i = 1$ , and a *suffix*, if  $j = |s|$ ). A prefix or suffix of  $s$  is called *proper* if it is not equal to  $s$ . A factor  $u$  may be equal to several substrings of  $s$ , referred to as *occurrences* of  $u$  in  $s$ . The occurrences of a given factor  $u$  are totally ordered by their positions in  $s$ , so we can speak about “leftmost” or “previous” occurrence. By  $u^k$  we denote the concatenation of  $k$  copies of string  $u$ . If  $k = 0$  we define  $u^k = \varepsilon$ .

<sup>2</sup> For example `gzip`, `p7zip`, `lz4`, and `snappy` all have the LZ factorization at their core.

<sup>3</sup> This is the non-overlapping version of the LZ factorization.

A string  $u$  over  $\Sigma$  is lexicographically smaller or equal than a string  $v$  (denoted by  $u \preceq v$ ) if either  $u$  is a prefix of  $v$  or  $u = xaw_1, v = xbw_2$  for some strings  $x, w_1, w_2$  and some letters  $a \prec b$ . In the latter case, we refer to this occurrence of  $a$  (resp., of  $b$ ) as the *mismatch* of  $u$  with  $v$  (resp., of  $v$  with  $u$ ). A string  $w$  is called a *Lyndon word* if  $w$  is lexicographically smaller than all its non-empty proper suffixes. The *Lyndon factorization* of a string  $s$  is its unique (see [10]) factorization  $s = f_1^{e_1} \cdots f_m^{e_m}$  such that each  $f_i$  is a Lyndon word,  $e_i \geq 1$ , and  $f_i \succ f_{i+1}$  for all  $1 \leq i < m$ . We call each  $f_i$  a *Lyndon factor* of  $s$ , and each  $F_i = f_i^{e_i}$  a *Lyndon run* of  $s$ . The size of the Lyndon factorization is  $m$ , the number of distinct Lyndon factors, or equivalently, the number of Lyndon runs.

The *non-overlapping LZ factorization* (see [26]) of a string  $s$  is its factorization  $s = p_1 \cdots p_z$  built left to right in a greedy way by the following rule: each new factor (also called an *LZ phrase*)  $p_i$  is either the leftmost occurrence of a letter in  $s$  or the longest prefix of  $p_i \cdots p_z$  which occurs in  $p_1 \cdots p_{i-1}$ .

### 3 Upper Bound

The aim of this section is to prove the following theorem.

► **Theorem 1.** *Every string  $s$  having Lyndon factorization  $s = f_1^{e_1} \cdots f_m^{e_m}$  and non-overlapping LZ factorization  $s = p_1 \cdots p_z$  satisfies  $m < 2z$ .*

Let us fix an arbitrary string  $s$  and relate all notation  $(f_i, e_i, F_i, p_i, m, z)$  to  $s$ . The main line of the proof is as follows. We identify occurrences of some factors in  $s$  that must contain a boundary between two LZ phrases. Non-overlapping occurrences contain different boundaries, so our aim is to prove the existence of more than  $m/2$  such occurrences. We start with two basic facts; the first one is obvious.

► **Lemma 2.** *For any strings  $u, v, w_1, w_2$ , the relation  $uw_1 \prec v \prec uw_2$  implies that  $u$  is a prefix of  $v$ .*

► **Lemma 3.** *The inequality  $j < i$  implies  $f_j \succ F_i$ .*

**Proof.** We prove that  $f_j \succ f_i^k$  for any  $k$ , arguing by induction on  $k$ . The base case  $k = 1$  follows from the definitions. Let  $f_j \succ f_i^{k-1}$ . In the case of mismatch,  $f_j \succ f_i^k$  holds trivially. Otherwise,  $f_j = f_i^{k-1}x$  for some  $x \neq \varepsilon$ . If  $x = f_i$  or  $x \prec f_i$ , then  $x \prec f_j$ , and so  $f_j$  is not a Lyndon word. Hence  $x \succ f_i$  and thus  $f_j = f_i^{k-1}x \succ f_i^k$ . Thus, the inductive step holds. ◀

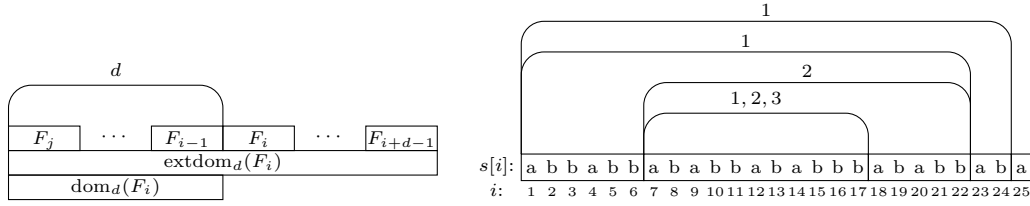
The next lemma locates the leftmost occurrences of the Lyndon runs and their products.

► **Lemma 4.** *Let  $d \geq 1$  and  $1 \leq i \leq m - d + 1$ , and assume that  $F_i F_{i+1} \cdots F_{i+d-1}$  has an occurrence to the left of the trivial one in  $s$ . Then:*

1. *The leftmost occurrence of  $F_i F_{i+1} \cdots F_{i+d-1}$  is a prefix of  $f_j$  for some  $j < i$ ;*
2.  *$F_i F_{i+1} \cdots F_{i+d-1}$  is a prefix of every  $f_k$  with  $j < k < i$ .*

**Proof.** (1) Let  $j$  be the smallest integer such that the leftmost occurrence of  $F_i F_{i+1} \cdots F_{i+d-1}$  in  $s$  overlaps  $F_j$ . Suppose first that the leftmost occurrence of  $F_i F_{i+1} \cdots F_{i+d-1}$  is not entirely contained inside a single occurrence of  $f_j$ . Then there exists a non-empty suffix  $u$  of  $f_j$  that is equal to some prefix of one of the factors  $f_i, \dots, f_{i+d-1}$ , say  $f_{i'}$ . We cannot have  $u = f_j$  because then  $f_j \preceq f_{i'}$  which is impossible since  $j < i'$ . Thus  $u$  must be a proper suffix of  $f_j$ . But then  $u \preceq f_{i'} \prec f_j$ , which contradicts  $f_j$  being a Lyndon word.

Suppose then that the leftmost occurrence of  $F_i F_{i+1} \cdots F_{i+d-1}$  in  $s$  is entirely contained inside  $f_j$  but is not its prefix, i.e.,  $f_j = vF_i F_{i+1} \cdots F_{i+d-1}w$  for some strings  $v \neq \varepsilon$  and  $w$ .



■ **Figure 1** Left: Graphical notation used to illustrate  $\text{dom}_d(F_i) = F_j \cdots F_{i-1}$ . Also shown is  $\text{extdom}_d(F_i) = F_j \cdots F_{i+d-1}$ . Right: all non-empty domains for the example string with the Lyndon factorization of size 5. Note that due to Lemma 6 there are no non-trivial intersections between domains.

Since  $f_j$  is a Lyndon word we have  $f_j \prec F_i F_{i+1} \cdots F_{i+d-1} w$ . Consider the position of the mismatch of  $F_i F_{i+1} \cdots F_{i+d-1} w$  with  $f_j$ . If the mismatch occurs inside  $F_i F_{i+1} \cdots F_{i+d-1}$ , we can write  $f_j = F_i \cdots F_{i'-1} f_{i'}^e x$  where  $i \leq i' < i + d$ ,  $0 \leq e < e_i$ , and  $x$  is a suffix of  $f_j$  that satisfies  $x \prec f_{i'} \prec f_j$ , which contradicts  $f_j$  being a Lyndon word. On the other hand, the mismatch inside  $w$  implies that  $f_j$  begins with  $F_i F_{i+1} \cdots F_{i+d-1}$ , contradicting the assumption that the inspected occurrence of  $F_i F_{i+1} \cdots F_{i+d-1}$  is the leftmost in  $s$ .

(2) We prove this part by induction on  $d$ . Let  $d = 1$ . By Lemma 3 we have  $f_j \succ f_k \succ F_i$ . Since  $f_j$  begins with  $F_i$  by statement 1, so does  $f_k$  by Lemma 2. Assume now that the claim holds for all  $d' < d$ . From the inductive assumption  $F_i$  and  $F_{i+1} \cdots F_{i+d-1}$  are both prefixes of  $f_k$ . Let  $y, y'$ , and  $z$  be such that  $f_j = F_i F_{i+1} \cdots F_{i+d-1} y$ ,  $f_k = F_{i+1} \cdots F_{i+d-1} y' = F_i z$ . We have  $j < k$  and thus  $f_k \prec f_j$  must hold which, since  $F_i$  is a prefix of both  $f_j$  and  $f_k$ , implies  $z \prec F_{i+1} \cdots F_{i+d-1} y$ . On the other hand, since  $f_k$  is a Lyndon word, we have  $f_k = F_{i+1} \cdots F_{i+d-1} y' \prec z$ . By Lemma 2,  $F_{i+1} \cdots F_{i+d-1} y' \prec z \prec F_{i+1} \cdots F_{i+d-1} y$  implies that  $F_{i+1} \cdots F_{i+d-1}$  is a prefix of  $z$  or equivalently that  $F_i F_{i+1} \cdots F_{i+d-1}$  is a prefix of  $f_k$ . ◀

### 3.1 Domains

Lemma 4 motivates the following definition.

► **Definition 5.** Let  $d \geq 1$  and  $1 \leq i \leq m - d + 1$ . We define the  $d$ -domain of Lyndon run  $F_i$  as the substring  $\text{dom}_d(F_i) = F_j F_{j+1} \cdots F_{i-1}$ ,  $j \leq i$  of  $s$ , where  $F_j$  is the Lyndon run (which exists by Lemma 4) starting at the same position as the leftmost occurrence of  $F_i F_{i+1} \cdots F_{i+d-1}$  in  $s$ . Note that if  $F_i F_{i+1} \cdots F_{i+d-1}$  does not have any occurrence to the left of the trivial one then  $\text{dom}_d(F_i) = \varepsilon$ . The integers  $d$  and  $i - j$  are called the *order* and *size* of the domain, respectively.

The *extended  $d$ -domain* of  $F_i$  is the substring  $\text{extdom}_d(F_i) = \text{dom}_d(F_i) F_i \cdots F_{i+d-1}$  of  $s$ .

Lemma 4 implies two easy properties of domains presented below as Lemma 6. These properties lead to a convenient graphical notation to illustrate domains (see Fig. 1).

- **Lemma 6.** Let  $\text{dom}_d(F_i) = F_j \cdots F_{i-1}$ ,  $j \leq i$ . Then:
  - For any  $d' > d$ ,  $\text{dom}_{d'}(F_i)$  is a suffix of  $\text{dom}_d(F_i)$ ;
  - For any  $d' \geq 1$ ,  $\text{dom}_{d'}(F_k)$  is a substring of  $\text{dom}_d(F_i)$  if  $j \leq k < i$ .

► **Definition 7.** Consider  $\text{dom}_d(F_i)$  for some  $d \geq 1$ ,  $1 \leq i \leq m - d + 1$ , and let  $\alpha = F_i \cdots F_{i+d-1}$ . We say that the leftmost occurrence of  $\alpha$  in  $s$  is *associated* with  $\text{dom}_d(F_i)$ .

For example, in Fig. 1,  $s[7..9]$  is associated with  $\text{dom}_2(s[23..24])$ ;  $s[7..17]$  is associated with  $\text{dom}_1(s[7..17])$  even though it is not shown, since  $\text{dom}_1(s[7..17]) = \varepsilon$ . Observe that due

to Lemma 6 this implies that  $\text{dom}_d(s[7..17]) = \varepsilon$  for any  $d > 1$ , and hence for example the substring of  $s$  associated with  $\text{dom}_2(s[7..17])$  is  $s[7..22]$ .

A substring  $s[i..i+k]$ ,  $k \geq 0$  is said to *contain an LZ phrase boundary* if some phrase of the LZ-factorization of  $s$  begins in one of the positions  $i, \dots, i+k$ . Clearly, non-overlapping substrings contain different phrase boundaries. Furthermore, if the substring of  $s$  does not have any occurrence to the left (in particular, if it is the leftmost occurrence of a single symbol), it contains an LZ phrase boundary, thus we obtain the following easy observation.

► **Lemma 8.** *Each substring associated with a domain contains an LZ phrase boundary.*

### 3.2 Tandem Domains

► **Definition 9.** Let  $d \geq 1$  and  $1 \leq i \leq m - d$ . A pair of domains  $\text{dom}_{d+1}(F_i)$ ,  $\text{dom}_d(F_{i+1})$  is called a *tandem domain* if  $\text{dom}_{d+1}(F_i) \cdot F_i = \text{dom}_d(F_{i+1})$  or, equivalently, if  $\text{extdom}_{d+1}(F_i) = \text{extdom}_d(F_{i+1})$ . Note that we permit  $\text{dom}_{d+1}(F_i) = \varepsilon$ .

For example,  $\text{dom}_3(s[18..22])$ ,  $\text{dom}_2(s[23..24])$  is a tandem domain in Fig. 1, because we have  $\text{extdom}_3(s[18..22]) = \text{extdom}_2(s[23..24]) = s[7..25]$ .

► **Definition 10.** Let  $\text{dom}_{d+1}(F_i)$ ,  $\text{dom}_d(F_{i+1})$  be a tandem domain. Since  $F_{i+1} \cdots F_{i+d}$  is a prefix of  $F_i$  by Lemma 4, we let  $F_i = F_{i+1} \cdots F_{i+d}x$ . The leftmost occurrence of  $F_i \cdots F_{i+d}$  in  $s$  can thus be written as  $F_{i+1} \cdots F_{i+d}xF_{i+1} \cdots F_{i+d}$ . We say that this particular occurrence of the factor  $xF_{i+1} \cdots F_{i+d}$  is *associated* with the tandem domain  $\text{dom}_{d+1}(F_i)$ ,  $\text{dom}_d(F_{i+1})$ .

► **Remark.** Note that the above definition permits  $\text{dom}_{d+1}(F_i) = \varepsilon$ . If  $\text{dom}_{d+1}(F_i) \neq \varepsilon$ , then  $\alpha$ , the substring of  $s$  associated with  $\text{dom}_{d+1}(F_i)$ ,  $\text{dom}_d(F_{i+1})$ , is (by Lemma 4) a substring of  $F_j$ , where  $F_j$ ,  $j < i$  is the leftmost Lyndon run inside  $\text{dom}_{d+1}(F_i)$ . Otherwise,  $\alpha$  overlaps at least two Lyndon runs. In both cases, however,  $\alpha$  is a substring of  $\text{extdom}_{d+1}(F_i)$ .

► **Lemma 11.** *Each substring associated with a tandem domain contains an LZ phrase boundary.*

**Proof.** Let  $\text{dom}_{d+1}(F_i)$ ,  $\text{dom}_d(F_{i+1})$  be a tandem domain and let  $u = xF_{i+1} \cdots F_{i+d}$  be the associated substring of  $s$ . Suppose to the contrary that  $u$  contains no LZ phrase boundaries. Then some LZ-phrase  $p_t$  contains  $u$  and the letter preceding  $u$ . Since we consider a non-overlapping LZ variant, the previous occurrence of  $p_t$  in  $s$  must be a substring of  $p_1 \cdots p_{t-1}$ . Note, however, that  $u$  is preceded in  $s$  by the leftmost occurrence of  $F_{i+1} \cdots F_{i+d}$ , which is the prefix of  $F_j$  (see Definition 10). Thus, the leftmost occurrence of  $u$  in  $s$  either immediately precedes the associated substring, or overlaps it, or coincides with it. This, however, rules out the possibility that the previous occurrence of  $p_t$  occurs in  $p_1 \cdots p_{t-1}$ , a contradiction. ◀

We say that a tandem domain  $\text{dom}_{d+1}(F_i)$ ,  $\text{dom}_d(F_{i+1})$  is *disjoint* from a tandem domain  $\text{dom}_{e+1}(F_k)$ ,  $\text{dom}_e(F_{k+1})$  if all  $i, i + 1, k, k + 1$  are different, i.e.,  $i + 1 < k$  or  $k + 1 < i$ .

► **Lemma 12.** *Substrings associated with disjoint tandem domains do not overlap each other.*

**Proof.** Let  $\text{dom}_{d+1}(F_i)$ ,  $\text{dom}_d(F_{i+1})$  and  $\text{dom}_{e+1}(F_k)$ ,  $\text{dom}_e(F_{k+1})$  be tandem domains called the  $d$ -tandem and  $e$ -tandem, respectively. Without the loss of generality let  $i + 1 < k$ .

Case 1:  $\text{dom}_{d+1}(F_i) \neq \varepsilon$  and  $\text{dom}_{e+1}(F_k) \neq \varepsilon$ . First observe that if the  $d$ -tandem and  $e$ -tandem begin with different Lyndon runs, then the associated substrings trivially do not overlap by the above Remark. Assume then that all considered domains start with  $F_j$ ,  $j < i$ . By Definition 10 we can write  $F_j$  as  $F_j = F_{i+1} \cdots F_{i+d}xF_{i+1} \cdots F_{i+d}y$ , where  $|F_{i+1} \cdots F_{i+d}x| = |F_i|$  and  $xF_{i+1} \cdots F_{i+d}$  is the substring of  $s$  associated with the  $d$ -tandem.

Similarly we have  $F_j = F_{k+1} \cdots F_{k+e} x' F_{k+1} \cdots F_{k+e} y'$  where  $|F_{k+1} \cdots F_{k+e} x'| = |F_k|$  and  $x' F_{k+1} \cdots F_{k+e}$  is the substring of  $s$  associated with the  $e$ -tandem. However, by Lemma 4,  $F_k \cdots F_{k+e}$  is a prefix of  $F_{i+1}$  and thus  $|F_{k+1} \cdots F_{k+e} x' F_{k+1} \cdots F_{k+e}| \leq |F_{i+1}|$ , i.e., the substring of  $s$  associated with the  $e$ -tandem is inside the prefix  $F_{i+1}$  of  $F_j$  and thus is on the left of the substring associated with the  $d$ -tandem.

Case 2:  $\text{dom}_{d+1}(F_i) = F_j \cdots F_{i-1}$ ,  $j < i$ , and  $\text{dom}_{e+1}(F_k) = \varepsilon$ . In this case the substring associated with the  $e$ -tandem begins in  $F_k$  by the above Remark and thus is on the right of the substring associated with the  $d$ -tandem.

Case 3:  $\text{dom}_{d+1}(F_i) = \varepsilon$  and  $\text{dom}_{e+1}(F_k) = \varepsilon$ . This is only possible if  $i + d < k$  since otherwise  $F_k$  (and thus also  $F_{k+1} \cdots F_{k+e}$ ) occurs in  $F_i$ , contradicting  $\text{dom}_e(F_{k+1}) = F_k$ . Then,  $\text{extdom}_{d+1}(F_i)$  does not overlap  $\text{extdom}_{e+1}(F_k)$ , and the claim holds by above Remark.

Case 4:  $\text{dom}_{d+1}(F_i) = \varepsilon$  and  $\text{dom}_{e+1}(F_k) = F_j \cdots F_{k-1}$ ,  $j < k$ . Then, the substring of  $s$  associated with  $e$ -tandem is a substring of  $F_j$ . If  $i > j$ , then clearly  $\text{extdom}_{d+1}(F_i)$  does not overlap  $F_j$ . On the other hand, if  $i < j$ , it must also hold  $i + d < j$  since otherwise  $F_j$  (and thus also  $F_k \cdots F_{k+e}$ ) occurs in  $F_i$ , contradicting  $\text{dom}_{e+1}(F_k) = F_j \cdots F_{k-1}$ , and thus again,  $\text{extdom}_{d+1}(F_i)$  does not overlap  $F_j$ . In both cases the Remark above implies the claim. Finally, if  $i = j$ , we must also have  $i + 1 < k$  from the assumption about the disjointness of  $d$ - and  $e$ -tandem. By Lemma 4 we can write  $F_i = F_{i+1} \cdots F_{i+d} x$ ,  $F_{i+1} = F_k \cdots F_{k+e} x'$  and hence also  $F_i \cdots F_{i+d} = F_k \cdots F_{k+e} x' F_{i+2} \cdots F_{i+d} x F_{i+1} \cdots F_{i+d}$ . In this decomposition, the substring associated with the  $e$ -tandem occurs inside the prefix  $F_k \cdots F_{k+e}$ , and the substring associated with the  $d$ -tandem is the suffix  $x F_{i+1} \cdots F_{i+d}$ , which proves the claim. ◀

### 3.3 Groups

We now generalize the concept of tandem domain.

► **Definition 13.** Let  $d \geq 1$ ,  $2 \leq p \leq m$ , and  $1 \leq i \leq m - d - p + 2$ . A set of  $p$  domains  $\text{dom}_{d+p-1}(F_i)$ ,  $\text{dom}_{d+p-2}(F_{i+1})$ ,  $\dots$ ,  $\text{dom}_d(F_{i+p-1})$  is called a  $p$ -group if for all  $t = 0, \dots, p-2$  the equality  $\text{dom}_{d+p-1-t}(F_{i+t}) \cdot F_{i+t} = \text{dom}_{d+p-2-t}(F_{i+t+1})$  holds or, equivalently,  $\text{extdom}_{d+p-1}(F_i) = \dots = \text{extdom}_d(F_{i+p-1})$ . Note that we permit  $\text{dom}_{d+p-1}(F_i) = \varepsilon$ .

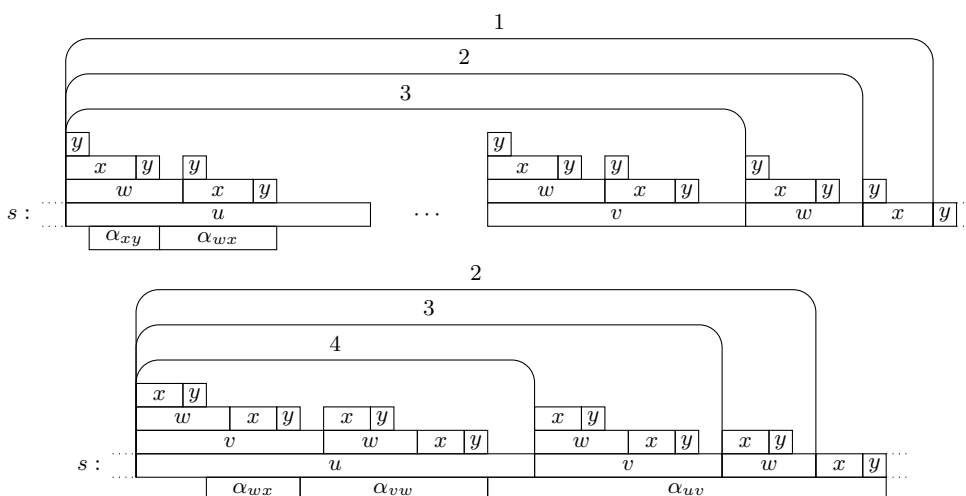
► **Lemma 14.** *Substrings associated with tandem domains from the same group do not overlap each other.*

**Proof.** Consider a  $p$ -group,  $p \geq 3$  and assume first that  $p = 3$ . By Lemma 4 we have  $F_i = F_{i+1} \cdots F_{i+d+1} x'$  and  $F_{i+1} = F_{i+2} \cdots F_{i+d+1} x$  for some words  $x'$  and  $x$ . We can thus write the leftmost occurrence of  $F_i \cdots F_{i+d+1}$  in  $s$  as  $F_{i+2} \cdots F_{i+d+1} x F_{i+2} \cdots F_{i+d+1} x' F_{i+1} \cdots F_{i+d+1}$ . It is easy to see that those occurrences of  $x F_{i+2} \cdots F_{i+d+1}$  and  $x' F_{i+1} \cdots F_{i+d+1}$  are associated with (resp.) tandem domains  $\text{dom}_{d+1}(F_{i+1})$ ,  $\text{dom}_d(F_{i+2})$  and  $\text{dom}_{d+2}(F_i)$ ,  $\text{dom}_{d+1}(F_{i+1})$ , and thus the claim holds.

For  $p > 3$  it suffices to consider all subgroups of size three, in left-to-right order, to verify that the substrings associated with all tandem domains occur in reversed order as a contiguous substring and thus no two substrings overlap each other. ◀

The above Lemma is illustrated in Fig. 2. It also motivates the following definition which generalizes the concept of associated substring from tandem domains to groups.

► **Definition 15.** Consider a  $p$ -group  $\text{dom}_{d+p-1}(F_i)$ ,  $\text{dom}_{d+p-2}(F_{i+1})$ ,  $\dots$ ,  $\text{dom}_d(F_{i+p-1})$  for some  $p \geq 2$ . From Lemma 4,  $F_{i+p-1} \cdots F_{i+p+d-2}$  is a prefix of  $F_i$ . Thus, the leftmost occurrence of  $F_i \cdots F_{i+p+d-2}$  in  $s$  can be written as  $F_{i+p-1} \cdots F_{i+p+d-2} x F_{i+1} \cdots F_{i+p+d-2}$ .



■ **Figure 2** Illustration of Lemma 14. In the examples  $u, v, w, x, y$  are Lyndon runs from the Lyndon factorization of  $s$ . The top figure shows a 3-group:  $\text{dom}_3(w) = u \cdots v$ ,  $\text{dom}_2(x) = u \cdots vw$ ,  $\text{dom}_1(y) = u \cdots vwx$ .  $\alpha_{wx}$  is a substring associated with the tandem domain  $\text{dom}_3(w)$ ,  $\text{dom}_2(x)$ , and  $\alpha_{xy}$  is a substring associated with the tandem domains  $\text{dom}_2(x)$ ,  $\text{dom}_1(y)$ . Observe that the substrings associated with tandem domains occur as a contiguous substring and in reverse order (compared to the order of the corresponding tandem domains in  $s$ ). The bottom figure shows a 4-group:  $\text{dom}_5(u) = \varepsilon$ ,  $\text{dom}_4(v) = u$ ,  $\text{dom}_3(w) = uv$ ,  $\text{dom}_2(x) = uvw$  and demonstrates the case when the leftmost domain in a group is empty.

We say that this particular occurrence of the substring  $x F_{i+1} \cdots F_{i+p+d-2}$  is *associated* with the  $p$ -group  $\text{dom}_{d+p-1}(F_i), \text{dom}_{d+p-2}(F_{i+1}), \dots, \text{dom}_d(F_{i+p-1})$ .

It is easy to derive a formal proof of the following Lemma from the proof of Lemma 14.

► **Lemma 16.** *The substring associated with a  $p$ -group is the concatenation, in reverse order, of the  $p - 1$  substrings associated with the tandem domains belonging to the  $p$ -group.*

Our consideration of groups culminates in the next two results.

► **Corollary 17.** *The substring associated with a  $p$ -group contains at least  $p - 1$  different LZ phrase boundaries.*

We say that a  $p$ -group  $\text{dom}_{d+p-1}(F_i), \dots, \text{dom}_d(F_{i+p-1})$  is *disjoint* from a  $p'$ -group  $\text{dom}_{d'+p'-1}(F_k), \dots, \text{dom}_{d'}(F_{k+p'-1})$  if  $i + p - 1 < k$  or  $k + p' - 1 < i$ . By combining Lemma 12 and Lemma 16 we obtain the following fact.

► **Lemma 18.** *Substrings associated with disjoint groups do not overlap.*

### 3.4 Subdomains

The concept of  $p$ -group does not easily extend to  $p = 1$ . If we simply define the 1-group as a single domain and extend the notion of groups to include 1-groups then Lemma 18 no longer holds (e.g. in Fig. 1 the substring associated with tandem domain  $\text{dom}_3(s[18..22])$ ,  $\text{dom}_2(s[23..24])$  is  $s[10..14]$  and the substring associated with domain  $\text{dom}_1(s[7..17])$  is  $s[7..17]$ ). Instead, we introduce a weaker lemma (Lemma 20) that also includes single domains.

► **Definition 19.** We say that a domain  $\text{dom}_e(F_k)$  is a *subdomain* of a domain  $\text{dom}_d(F_i) = F_j \cdots F_{i-1}$ ,  $j \leq i$  if  $k = i$  and  $e = d$  (i.e., the domain is its own subdomain), or  $j \leq k < i$  and  $\text{extdom}_e(F_k)$  is a substring of  $\text{extdom}_d(F_i)$  (or equivalently, if  $k + e \leq i + d$ ). In other words,  $F_k$  has to be one of the Lyndon runs among  $F_j, \dots, F_{i-1}$  and the extended domain of  $F_k$  cannot extend (to the right) beyond the extended domain of  $F_i$ .

► **Lemma 20.** Consider a tandem domain  $\text{dom}_{e+1}(F_k), \text{dom}_e(F_{k+1})$  such that  $\text{dom}_{e+1}(F_k)$  and  $\text{dom}_e(F_{k+1})$  are subdomains of  $\text{dom}_d(F_i)$ . Then, the substring associated with the tandem domain  $\text{dom}_{e+1}(F_k), \text{dom}_e(F_{k+1})$  does not overlap the substring associated with  $\text{dom}_d(F_i)$ .

**Proof.** First, observe that in order for a tandem domain consisting of two subdomains to exist,  $\text{dom}_d(F_i)$  has to be non-empty. Thus, let  $\text{dom}_d(F_i) = F_j \cdots F_{i-1}$  for some  $j < i$ . This implies (Lemma 4) that the substring associated with  $\text{dom}_d(F_i)$  is a prefix of  $F_j$ .

Assume first that  $\text{dom}_{e+1}(F_k) = F_{j'} \cdots F_{k-1}$ ,  $j < j' \leq k$ . The substring associated with the tandem domain is a substring of  $\text{extdom}_{e+1}(F_k)$  thus it trivially does not overlap  $F_j$ .

Assume then that  $\text{dom}_{e+1}(F_k) = F_j \cdots F_{k-1}$ . If  $k + 1 < i$  then by Lemma 4,  $F_i \cdots F_{i+d-1}$  is a prefix of  $F_{k+1}$ . By Definition 10 the leftmost occurrence of  $F_k \cdots F_{k+e}$  in  $s$  can be written as  $F_{k+1} \cdots F_{k+e} x F_{k+1} \cdots F_{k+e}$ . Thus clearly the leftmost occurrence of  $F_i \cdots F_{i+d-1}$  (associated with  $\text{dom}_k(F_i)$ ) occurs in a prefix  $F_{k+1}$  not overlapped by  $x F_{k+1} \cdots F_{k+e}$  (which is a substring associated with the tandem domain).

The remaining case is when  $k + 1 = i$ . Then by Definition 19 we must have  $e = d$  and again the claim holds easily from Definition 10. ◀

For any domain  $\text{dom}_d(F_i) = F_j \cdots F_{i-1}$ ,  $j < i$  we define the set of *canonical subdomains* as follows. Consider the following procedure. Initialize the set of canonical subdomains to contain  $\text{dom}_d(F_i)$ . Then initialize  $\delta = d$  and start scanning the Lyndon runs  $F_j, \dots, F_{i-1}$  right-to-left. When scanning  $F_t$  we check if  $\text{dom}_{\delta+1}(F_t) = F_j \cdots F_{t-1}$ .

- If yes, we include  $\text{dom}_{\delta+1}(F_t)$  into the set, increment  $\delta$  and continue scanning from  $F_{t-1}$ .
- Otherwise, i.e., if  $\text{dom}_{\delta+1}(F_t) = F_{j'} \cdots F_{t-1}$  for some  $j' > j$ , we include the domain  $\text{dom}_{\delta+1}(F_t)$  into the set. Then we set  $\delta = 0$  and continue scanning from  $F_{j'-1}$ . All domains that were included into the set of canonical subdomains in this case are called *loose* subdomains.

See Fig. 3 for an example. The above procedure simply greedily constructs groups of domains, and whenever the candidate for the next domain in the current group does not have a domain that starts with  $F_j$ , we terminate the current group, add the loose subdomain into the set and continue building groups starting with the next Lyndon run outside the (just included) loose subdomain.

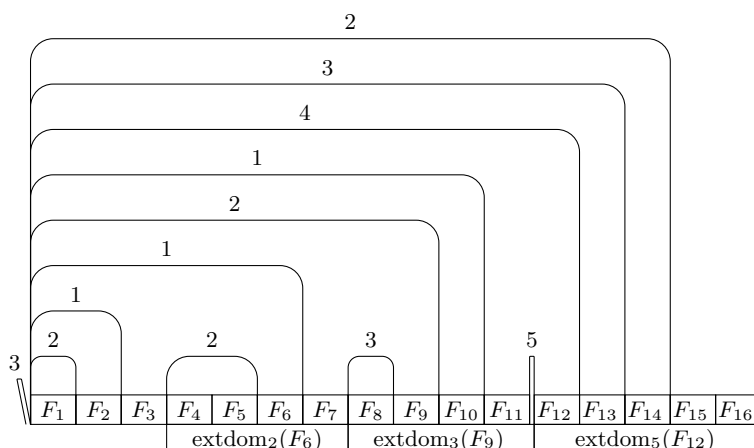
Note that the current group can be terminated when containing just one domain, so it is not a group in this case. Hence we call the resulting sequences of non-loose domains *clusters*, i.e., a cluster is either a single domain, or a  $p$ -group,  $p \geq 2$ . Note also that during the construction we may encounter more than one loose subdomain in a row, so clusters and loose subdomains do not necessarily alternate, but no two clusters occur consecutively.

Finally, observe that the sequence of clusters and loose subdomains always ends with a cluster (possibly of size one) containing  $\text{dom}_{d'}(F_j)$  for some  $d'$  ( $d' = 3$  for the example in Fig. 3), since  $\text{dom}_{d'}(F_j) = \varepsilon$  for all  $d'$ .

### 3.5 Proof of the Main Theorem

We are now ready to prove the key Lemma of the proof. Recall that the size of  $\text{dom}_d(F_i) = F_j \cdots F_{i-1}$ ,  $j \leq i$  is defined as  $i - j$ .





■ **Figure 3** An example showing the set of canonical subdomains of  $\text{dom}_2(F_{15})$ . Using notation from Lemma 21, the set has  $p = 4$  clusters of size (left-to-right):  $\ell_1 = 3, \ell_2 = 1, \ell_3 = 2, \ell_4 = 3$ , and  $t = 3$  loose subdomains:  $\text{dom}_2(F_6) = F_4F_5$ ,  $\text{dom}_3(F_9) = F_8$ ,  $\text{dom}_5(F_{12}) = \varepsilon$  of size  $k_1 = 2, k_2 = 1, k_3 = 0$ . Note how the extended domains of loose subdomains do not overlap each other. Furthermore, note that  $\text{extdom}_2(F_{15}) = F_1 \cdots F_{16}$  can be factorized as  $F_1 \cdots F_{\ell_1}$  concatenated with the extended domains. By Corollary 17 and Lemmas 18 and 20,  $F_1 \cdots F_{\ell_1}$  contains  $1 + \sum_{h=1}^p (\ell_h - 1) = 6$  LZ phrase boundaries, while the extended domains  $\text{extdom}_2(F_6)$ ,  $\text{extdom}_3(F_9)$ ,  $\text{extdom}_5(F_{12})$  contain  $\sum_{h=1}^t (\lceil k_h/2 \rceil + 1) = 5$  LZ phrase boundaries by Lemma 21.

► **Lemma 21.** *Let  $\text{dom}_d(F_i)$  be a domain of size  $k \geq 0$ . Then  $\text{extdom}_d(F_i)$  contains at least  $\lceil k/2 \rceil + 1$  different LZ phrase boundaries.*

**Proof.** Let  $\text{dom}_d(F_i) = F_j \cdots F_{i-1}$ ,  $j \leq i$  and  $k = i - j$ . The proof is by induction on  $k$ . For  $k = 0$ ,  $\text{extdom}_d(F_i)$  is the substring of  $s$  associated with  $\text{dom}_d(F_i)$  (see Definition 7) and thus by Lemma 8,  $\text{extdom}_d(F_i)$  contains at least one LZ phrase boundary.

Let  $k > 0$  and assume now that the claim holds for all smaller  $k$ . Consider the set  $\mathcal{C}_{i,d}$  of canonical subdomains of  $\text{dom}_d(F_i)$ . If  $\mathcal{C}_{i,d}$  contains no loose subdomain, it consists of a single cluster which is a  $(k + 1)$ -group. By Corollary 17, the substring associated with this group contains  $k$  phrase boundaries; by Lemma 20, one more boundary is provided by the domain  $\text{dom}_d(F_i)$  itself. We have  $1 + k \geq 1 + \lceil k/2 \rceil$ , which concludes the proof of this case.

For the rest of the proof assume that  $\mathcal{C}_{i,d}$  contains  $t \geq 1$  loose subdomains, denoted, left to right, by  $\text{dom}_{d_1}(F_{i_1}), \dots, \text{dom}_{d_t}(F_{i_t})$ . Note that  $d_t > d$ . Let  $k_h$  be the size of  $\text{dom}_{d_h}(F_{i_h})$ ,  $h = 1, \dots, t$ . Further, let  $\ell \geq 1$  be the size of the leftmost cluster (the one that contains some domain of  $F_j$ ). By the construction of the canonical set we have

$$\text{extdom}_d(F_i) = F_j \cdots F_{j+\ell-1} \text{extdom}_{d_1}(F_{i_1}) \text{extdom}_{d_2}(F_{i_2}) \cdots \text{extdom}_{d_t}(F_{i_t}). \tag{1}$$

Both clusters and loose subdomains contribute some number of LZ phrase boundaries into their total. The boundaries contributed by clusters are all different by Lemma 18; let  $S$  be their number. These boundaries are also different from the boundary inside the substring associated with  $\text{dom}_d(F_i)$  by Lemma 20. Furthermore, it is easy to see from the proof of Lemma 20 that all these phrase boundaries are located inside  $F_j \cdots F_{j+\ell-1}$ . The number of phrase boundaries inside the extended domains of loose subdomains can be estimated by the inductive assumption (by Eq. 1, these external domains do not overlap each other or

## 45:10 On the Size of Lempel-Ziv and Lyndon Factorizations

$F_j \cdots F_{j+\ell-1}$ ). So we obtain that  $\text{extdom}_d(F_i)$  contains at least

$$1 + \sum_{h=1}^t \left( \left\lceil \frac{k_h}{2} \right\rceil + 1 \right) + S \quad (2)$$

different LZ phrase boundaries. Let us evaluate  $\sum_{h=1}^t k_h$ . By the construction, a loose  $d_h$ -subdomain is followed by exactly  $d_h$  Lyndon runs which are outside loose subdomains; then another loose subdomain follows (cf. Fig. 3). The only exception is the rightmost loose subdomain, which is followed by  $d_t - d$  Lyndon runs outside loose subdomains (note that we only count Lyndon runs inside  $\text{dom}_d(F_i)$ ). Then

$$\sum_{h=1}^t k_h = k - \ell - \sum_{h=1}^t d_h + d. \quad (3)$$

Next we evaluate  $S$ . By Corollary 17, a cluster of size  $r$  contributes  $r - 1$  phrase boundaries. Then the leftmost (resp., rightmost) cluster contributes  $\ell - 1$  (resp.,  $d_t - d - 1$ ) boundaries. Each of the remaining clusters is preceded by a loose  $d_h$ -subdomain, where  $d_h > 1$ , and contributes  $d_h - 2$  boundaries. Using Knuth's notation  $[predicate]$  for the numerical value (0 or 1) of the predicate in brackets, we can write

$$S = \ell - 1 + \sum_{h=1}^t d_h - t - d - \sum_{h=1}^{t-1} [d_h > 1]. \quad (4)$$

Finally, we estimate the number in Eq. 2 using Eq. 3 and Eq. 4:

$$\begin{aligned} 1 + \sum_{h=1}^t \left( \left\lceil \frac{k_h}{2} \right\rceil + 1 \right) + S &\geq 1 + t + \frac{k - \ell - \sum_{h=1}^t d_h + d}{2} + \ell - 1 + \sum_{h=1}^t d_h - t - d - \sum_{h=1}^{t-1} [d_h > 1] \\ &= \frac{k}{2} + \frac{\ell}{2} + \sum_{h=1}^t \frac{d_h}{2} - \frac{d}{2} - \sum_{h=1}^{t-1} [d_h > 1] = \frac{\ell + d_t - d}{2} + \frac{k}{2} + \sum_{h=1}^{t-1} \left( \frac{d_h}{2} - [d_h > 1] \right) \geq 1 + \frac{k}{2}. \end{aligned}$$

The obtained lower bound for an integer can be rounded up to  $1 + \lceil k/2 \rceil$ , as required. ◀

Using the above Lemma we can finally prove the main Theorem.

**Proof of Theorem 1.** Partition the string  $s$  into extended domains as follows: take the string  $s'$  such that  $s = s' \cdot \text{extdom}_1(F_m)$  and partition  $s'$  recursively to get

$$s = \text{extdom}_1(F_{i_1}) \cdots \text{extdom}_1(F_{i_t}), \text{ where } i_t = m.$$

By Lemma 21, each extended domain  $\text{extdom}_1(F_{i_h})$  contains at least  $\lceil k_h/2 \rceil + 1$  phrase boundaries, where  $k_h$  is the size of the domain  $\text{dom}_1(F_{i_h})$ . Clearly,  $\sum_{h=1}^t k_h = m - t$ ; hence the total number  $z$  of the boundaries satisfies

$$z \geq \sum_{h=1}^t \left( \left\lceil \frac{k_h}{2} \right\rceil + 1 \right) \geq \left\lceil \frac{m-t}{2} \right\rceil + t = \left\lceil \frac{m+t}{2} \right\rceil > \frac{m}{2},$$

as required. ◀

## 4 Lower Bound

The upper bound on the number of factors in the Lyndon factorization of a string, given in of Theorem 1, is supported by the following lower bound. Consider a string  $s_k = B_0 \cdots B_k a$ ,  $k \geq 0$ , where:

$$\begin{aligned} B_0 &= b, \\ B_1 &= ab, \\ B_2 &= a^2 b a b a^2 b, \\ &\dots \\ B_k &= (a^k b a^1 b) \cdots (a^k b a^{k-1} b) a^k b. \end{aligned}$$

For example,  $s_3 = (b)(ab)(a^2 b a b a^2 b)(a^3 b a b a^3 b a^2 b a^3 b)(a)$ .

► **Theorem 22.** *Let  $f_1 \cdots f_{m_k}$  and  $p_1 \cdots p_{z_k}$  be the Lyndon factorization and the non-overlapping LZ factorization of the string  $s_k$ ,  $k \geq 2$ . Then  $m_k = k^2/2 + k/2 + 2$ ,  $z_k = k^2/2 - k/2 + 4$ , and thus  $m_k = z_k + \Theta(\sqrt{z_k})$ .*

**Proof.** First we count Lyndon factors. All factors will be different, so their number coincides with the number of Lyndon runs. By the definition of Lyndon factorization, the block  $B_i$  ( $0 < i \leq k$ ) is factorized into  $i$  Lyndon factors:

$$B_i = a^i b a^1 b \cdot a^i b a^2 b \cdots a^i b a^{i-1} b \cdot a^i b. \tag{5}$$

For any suffix  $u$  of  $B_0 \cdots B_{i-1}$  and any prefix  $v$  of  $B_i$ ,  $u \succ v$  holds since  $a^i$  is a prefix of  $B_i$  and this is the leftmost occurrence of  $a^i$ . Thus there is no Lyndon word that begins in  $B_0 \cdots B_{i-1}$  and ends in  $B_i$ . This implies that the factorization of  $s_k$  is the concatenation of the first  $b$ , then  $k$  factorizations Eq. 5, and the final  $a$ ,  $k^2/2 + k/2 + 2$  factors in total.

Let  $LZ(s_k)$  denote the LZ factorization of  $s_k$ . The size of  $LZ(s_2) = b \cdot a \cdot ba \cdot aba \cdot baaba$  is 5. For  $k \geq 3$ , we prove by induction that

$$LZ(s_k) = LZ(s_{k-1}) \cdot a^{k-1} b a b a^{k-1} \cdot a b a^2 b a^{k-1} \cdots a b a^{k-2} b a^{k-1} \cdot a b a^{k-1} b a^k b a. \tag{6}$$

For  $k = 3$  we have  $LZ(s_3) = LZ(s_2) \cdot a a b a b a a \cdot a b a b a a a b a$  and thus the claim holds. If  $k > 3$ , by the inductive hypothesis the last phrase in  $LZ(s_{k-1})$  is  $p = a b a^{k-2} b a^{k-1} b a$ . The factor  $p$  has only one previous occurrence: it occurs at the boundary between  $B_{k-2}$  and  $B_{k-1}$ , followed by  $b$ . So,  $p$  remains a phrase in  $LZ(s_k)$ . Each of subsequent  $k - 2$  phrases of Eq. 6 also has a single previous occurrence (inside  $B_{k-1}$ ), and this occurrence is followed by  $b$  because  $B_{k-1}$  has no factor  $a^k$ . Thus, Eq. 6 correctly represents  $LZ(s_k)$ . Direct computation now gives  $z_k = k^2/2 - k/2 + 4$ . ◀

---

### References

- 1 Golnaz Badkobeh, Hideo Bannai, Keisuke Goto, Tomohiro I, Costas S. Iliopoulos, Shunsuke Inenaga, Simon J. Puglisi, and Shiho Sugimoto. Closed factorization. *Discrete Appl. Math.*, 212:23–29, 2016.
- 2 Hideo Bannai, Travis Gagie, Shunsuke Inenaga, Juha Kärkkäinen, Dominik Kempa, Marcin Piątkowski, Simon J. Puglisi, and Shiho Sugimoto. Diverse palindromic factorization is NP-complete. In *Proceedings of the 19th International Conference on Developments in Language Theory (DLT)*, pages 85–96. Springer, 2015.
- 3 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The runs theorem. *arXiv*, abs/1406.0263, 2014.

- 4 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. A new characterization of maximal repetitions by Lyndon trees. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 562–571. SIAM, 2015.
- 5 Djamel Belazzougui, Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Alberto Ordóñez Pereira, Simon J. Puglisi, and Yasuo Tabei. Queries on LZ-bounded encodings. In *Proceedings of the 2015 Data Compression Conference (DCC)*, pages 83–92. IEEE, 2015.
- 6 Srečko Brlek, Jacques-Olivier Lachaud, Xavier Provençal, and Christophe Reutenauer. Lyndon + Christoffel = digitally convex. *Pattern Recogn.*, 42(10):2239–2246, 2009.
- 7 Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2), 2008.
- 8 Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *IEEE Trans. Information Theory*, 51(7):2554–2576, 2005.
- 9 Marc Chemillier. Periodic musical sequences and Lyndon words. *Soft Comput.*, 8(9):611–616, 2004.
- 10 Kuo-Tsai Chen, Ralph H. Fox, and Roger C. Lyndon. Free differential calculus, IV. The quotient groups of the lower central series. *Ann. Math.*, 68:81–95, 1958.
- 11 Yoann Dieudonné and Franck Petit. Circle formation of weak robots and Lyndon words. *Inf. Process. Lett.*, 101(4):156–162, 2007.
- 12 Gabriele Fici, Travis Gagie, Juha Kärkkäinen, and Dominik Kempa. A subquadratic algorithm for minimum palindromic factorization. *J. Discrete Algorithms*, 28:41–48, 2014.
- 13 Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. LZ77-based self-indexing with faster pattern matching. In *Proceedings of the 11th Latin American Theoretical Informatics Symposium (LATIN)*, pages 731–742. Springer, 2014.
- 14 Joseph Yossi Gil and David Allen Scott. A bijective string sorting transform. *arXiv*, abs/1201.3077, 2012.
- 15 David Hill, George Melvin, and Damien Mondragon. Representations of quiver Hecke algebras via Lyndon bases. *J. Pure Appl. Algebr.*, 216:1052–1079, 2012.
- 16 Christopher Hoobin, Simon J. Puglisi, and Justin Zobel. Relative Lempel-Ziv factorization for efficient storage and retrieval of web collections. *PVLDB*, 5(3):265–273, 2011.
- 17 Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 596–604. IEEE, 1999.
- 18 Manfred Kufleitner. On bijective variants of the Burrows-Wheeler transform. In *Proceedings of the 2009 Prague Stringology Conference (PSC)*, pages 65–79. Czech Technical University in Prague, 2009.
- 19 Pierre Lalonde and Arun Ram. Standard Lyndon bases of Lie algebras and enveloping algebras. *Transactions of the American Mathematical Society*, 347:1821–1830, 1995.
- 20 M. Lothaire. *Combinatorics on Words*. Cambridge University Press, 1997.
- 21 Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. Suffix array and Lyndon factorization of a text. *J. Discrete Algorithms*, 28:2–8, 2014.
- 22 Yoshiaki Matsuoka, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda, and Florin Manea. Factorizing a string into squares in linear time. In *Proceedings of the 27th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 27:1–27:12. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016.
- 23 Marcin Mucha. Lyndon words and short superstrings. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 958–972. SIAM, 2013.

- 24 Wojciech Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211–222, 2003.
- 25 I Tomohiro, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Faster Lyndon factorization algorithms for SLP and LZ78 compressed text. *Theor. Comput. Sci.*, 656:215–224, 2016.
- 26 Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, 23(3):337–343, 1977.



# Voting and Bribing in Single-Exponential Time

Dušan Knop<sup>\*1</sup>, Martin Koutecký<sup>†2</sup>, and Matthias Mnich<sup>‡3</sup>

1 Dept. of Applied Mathematics, Charles University, Prague, Czech Republic  
knop@kam.mff.cuni.cz

2 Dept. of Applied Mathematics, Charles University, Prague, Czech Republic  
koutecky@kam.mff.cuni.cz

3 Maastricht University, Dept. of Quantitative Economics, Maastricht,  
The Netherlands; and  
Universität Bonn, Institut für Informatik, Bonn, Germany  
mmnich@uni-bonn.de

---

## Abstract

We introduce a general problem about bribery in voting systems. In the  $\mathcal{R}$ -MULTI-BRIBERY problem, the goal is to bribe a set of voters at minimum cost such that a desired candidate wins the manipulated election under the voting rule  $\mathcal{R}$ . Voters assign prices for withdrawing their vote, for swapping the positions of two consecutive candidates in their preference order, and for perturbing their approval count for a candidate.

As our main result, we show that  $\mathcal{R}$ -MULTI-BRIBERY is fixed-parameter tractable parameterized by the number of candidates for many natural voting rules  $\mathcal{R}$ , including Kemeny rule, all scoring protocols, maximin rule, Bucklin rule, fallback rule, SP-AV, and any C1 rule. In particular, our result resolves the parameterized  $\mathcal{R}$ -SWAP BRIBERY for all those voting rules, thereby solving a long-standing open problem and “Challenge #2” of the 9 Challenges in computational social choice by Bredereck et al.

Further, our algorithm runs in single-exponential time for arbitrary cost; it thus improves the earlier double-exponential time algorithm by Dorn and Schlotter that is restricted to the unit-cost case for all scoring protocols, the maximin rule, and Bucklin rule.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, G.2 Discrete Mathematics, J.4 Social and Behavioral Sciences

**Keywords and phrases** Parameterized algorithm, swap bribery, n-fold integer programming

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.46

## 1 Introduction

In this work we address algorithmic problems from the area of voting and bribing. In these problems, we are given as input an election, which consists of set  $C$  of candidates and a set  $V$  of voters  $v$ , each of which is equipped with a total order  $\prec_v$  indicating their preferences over the candidates. Further, we have a fixed voting rule  $\mathcal{R}$  (that is not part of the input), which determines how the orders of the voters are aggregated to determine the winner(s) of the election among the candidates. Popular examples of voting rules  $\mathcal{R}$  include “scoring protocols” like *plurality* – where the candidate(s) ranked first by a majority of voters win(s) – or the Borda rule, where each candidate receives  $|C| - i$  points from being

---

\* D. K. was supported by project CE-ITI P202/12/G061 of GA ČR and project 1784214 of GA UK.

† M. K. was supported by project 14-10003S of GA ČR and projects 1784214 and 338216 of GA UK.

‡ M. M. was supported by ERC Starting Grant 306465 (BeyondWorstCase).



© Dušan Knop, Martin Koutecký, and Matthias Mnich;  
licensed under Creative Commons License CC-BY

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 46; pp. 46:1–46:14

Leibniz International Proceedings in Informatics



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

■ **Table 1**  $\mathcal{R}$ -MULTI-BRIBERY generalizes several studied bribery problems. For Kemeny rule, no previous results are known to us. Also, XP denotes an algorithm with run time  $n^{f(|C|)}$  and FPT-AS a fixed-parameter approximation scheme.

Problem	Specialization of $\mathcal{R}$ -MULTI-BRIBERY	Previous best result (except Kemeny rule)
$\mathcal{R}$ - $\$$ BRIBERY	$\pi_i = 0, \alpha_i = \infty, a_i = d_i = \infty$	$2^{2^{O( C )}} \cdot n^{O(1)}$ [9]
$\mathcal{R}$ -MANIPULATION	$\iota_i = 0$ for $i \in S$ , $\iota_i = \infty$ for $i \notin S$	$2^{2^{O( C )}} \cdot n^{O(1)}$ [9]
$\mathcal{R}$ -CCAV/ $\mathcal{R}$ -CCDV	$\iota_i = 0, \pi_i = \alpha_i = \infty$	$2^{2^{O( C )}} \cdot n^{O(1)}$ [9]
$\mathcal{R}$ -SWAP BRIBERY	$\alpha_i = \infty, a_i = d_i = \infty, \iota_i = 0$	$2^{2^{O( C )}} \cdot n^{O(1)}$ , unit cost [13]
$\mathcal{R}$ -SHIFT BRIBERY	$\mathcal{R}$ -SWAP BRIBERY with $\pi_i(a, b) = \infty$ for $a, b \neq c_1$	XP, arbitrary cost, FPT-AS, restricted cost [10]
$\mathcal{R}$ -SUPPORT BRIB.	$\pi_i = a_i = d_i = \infty, \iota_i = 0$	NP-c [26]
$\mathcal{R}$ -MIXED BRIB.	$a_i = d_i = \infty, \iota_i = 0$	NP-c [14]
$\mathcal{R}$ -EXTENSION BRIB.	$\pi_i(a, b) = 0$ if $\text{rank}_i(a, b) > \iota_i$ , else $\pi_i(a, b) = \infty, a_i = d_i = \infty, \iota_i = 0$	NP-c [2]
$\mathcal{R}$ -POSSIBLE WIN.	reduce to $\mathcal{R}$ -SWAP BRIBERY [14, Thm 2]	$2^{2^{O( C )}} \cdot n^{O(1)}$ [4]
DODGSON SCORE	Condorcet-SWAP BRIBERY with $\pi_i = 1$	$2^{2^{O( C )}} \cdot n^{O(1)}$ [1]
YOUNG SCORE	$\mathcal{R}$ -CCDV with $\mathcal{R} = \text{Condorcet}, d_i = 1$	$2^{2^{O( C )}} \cdot n^{O(1)}$ [27]

ranked  $i$ -th by a voter and the candidate with most points wins; and the Copeland rule, which orders candidates by their number of pairwise victories minus their number of pairwise defeats. The goal is to *manipulate* the given election  $(C, V)$  by some actions  $\Theta$  in such a way that a designated candidate  $c_1 \in C$  wins the perturbed election  $(C, V)^\Theta$  under the fixed voting rule  $\mathcal{R}$ . Such manipulation problems model various real-life issues, such as actual bribery, or campaign management, or post-election checks, as in destructive bribery (known as margin of victory). Manipulation is performed by the actions of *swapping* the position of two adjacent candidates in the preference order of some voter, by *support changes* that perturb the approval count of a voter, and *control changes* that (de)activate some voters. The algorithmic problem is to achieve the goal by performing the most cost-efficient actions. To measure cost of swaps, we consider the model introduced by Elkind et al. [14] where each voter may assign different prices for swapping two consecutive candidates in their preference order; this captures the notion of small changes and comprises the preferences of the voters. We additionally allow voter-individual cost for support changes and control changes. We call this the  $\mathcal{R}$ -MULTI-BRIBERY problem.

Various special cases of the  $\mathcal{R}$ -MULTI-BRIBERY problem have been studied in the literature; see Table 1 for an overview which problems are captured by  $\mathcal{R}$ -MULTI-BRIBERY.

For instance, Faliszewski et al. [18] introduced the  $\mathcal{R}$ -SWAP BRIBERY problem, where only swaps are permitted. Of particular interest has been to understand the computational complexity such problems with respect to the number  $|C|$  of candidates [1, 4, 7, 8, 9, 10, 11, 13, 19]. A common outcome are *fixed-parameter algorithms* that find an optimal solution of each instance  $I$  in time  $f(|C|) \cdot |I|^{O(1)}$  for some function  $f$ ; for example, Dorn and Schlotter [13] show how to solve  $\mathcal{R}$ -SWAP BRIBERY with unit costs in time  $2^{2^{O(|C|)}} \cdot |I|^{O(1)}$  for so-called linearly describable voting rules  $\mathcal{R}$ . In general, the function  $f$  grows quite fast, often double-exponential in  $|C|$  which stems from solving a certain integer linear program (ILP) at some point of the algorithm. This observation led Brederick et al. [6] to put forward

<sup>1</sup> Brederick et al. [7] pointed out that the algorithm by Dorn and Schlotter only works for unit costs.



the following “Challenge #1”, as part of their “Nine Research Challenges in Social Choice”:

*Many [FPT results in computational social choice] rely on a deep result from combinatorial optimization due to Lenstra [that] is mainly of theoretical interest; this may render corresponding fixed-parameter tractability results to be of classification nature only. Can the mentioned ILP-based [...] results be replaced by direct combinatorial [...] fixed-parameter algorithms?*

Another downside of the “ILP-based approach” is that it inherently treats voters not as individuals, but as groups which share preferences. This makes it difficult to obtain algorithms where voters from the same group differ in some way, such as by the cost of bribing them. Their “Challenge #2” thus reads:

*[T]here is a huge difference between [...] problems, where each voter has unit cost for being bribed, and the other flavors of bribery, where each voter has individually specified price [...] and it is not known if they are in FPT or hard for W[1]. What is the exact parameterized complexity of the  $\mathcal{R}$ -SWAP BRIBERY and  $\mathcal{R}$ -SHIFT BRIBERY parameterized by the number of candidates, for each voting rule  $\mathcal{R}$ ?*

**Our contribution.** Our main result is a fixed-parameter algorithm for  $\mathcal{R}$ -MULTI-BRIBERY parameterized by the number of candidates, for many fundamental voting rules  $\mathcal{R}$ . In particular, our algorithm works for voter-dependent cost functions, and it runs in time that is only single-exponential in  $|C|$ .

► **Theorem 1.**  $\mathcal{R}$ -MULTI-BRIBERY is fixed-parameter tractable parameterized by the number of candidates, and can be solved in time

- $2^{O(|C|^6 \log |C|)} \cdot n^3$  if  $\mathcal{R}$  is a scoring protocol, any C1 rule, or SP-AV,
- $2^{O(|C|^6 \log |C|)} \cdot n^4$  if  $\mathcal{R}$  is the maximin, Bucklin or fallback rule, and
- $2^{O(|C|!^6)} \cdot n^3$  if  $\mathcal{R}$  is the Kemeny rule.

We argued  $\mathcal{R}$ -MULTI-BRIBERY generalizes many well-studied voting and bribing problems, parameterized by the number of candidates. A direct corollary of Theorem 1 is:

► **Corollary 2.** Let  $\mathcal{R}$  be a scoring protocol, a C1 rule, the maximin rule, the Bucklin rule, the SP-AV rule, the fallback rule, or Kemeny rule. Then  $\mathcal{R}$ -SWAP BRIBERY with arbitrary cost is fixed-parameter tractable parameterized by the number  $|C|$  of candidates.

This solves “Challenge #2” by Bredereck et al. [6]. In particular, for scoring protocols, maximin rule and Bucklin rule, Corollary 2 extends and improves an algorithm by Dorn and Schlotter [13] that is restricted to the unit-cost case of  $\mathcal{R}$ -SWAP BRIBERY, and requires double-exponential run time  $2^{2^{O(|C|)}} \cdot n^{O(1)}$ .

Furthermore, we avoid using Lenstra’s algorithm for solving ILPs with bounded number of variables, and thereby achieve the exponential improvements over previous run times for  $\mathcal{R}$ -SWAP BRIBERY. This way, we substantially contribute towards resolving “Challenge #1” by Bredereck et al.

We remark that it is unclear (cf. [19, p. 338]) if the Kemeny rule can be described by linear inequalities as defined by Dorn and Schlotter [13]; even if it does, ours is the first fixed-parameter algorithm for  $\mathcal{R}$ -SWAP BRIBERY under the Kemeny rule, as Dorn and Schlotter’s algorithm only applies to the unit-cost case.

Another corollary of Theorem 1 is the following:

► **Corollary 3.**  $\mathcal{R}$ -SHIFT BRIBERY is fixed-parameter tractable parameterized by the number of candidates, for  $\mathcal{R}$  being the Borda rule, the maximin rule and the Copeland<sup>α</sup> rule.

This way, we simultaneously improve the fixed-parameter algorithm by Dorn and Schlotter [13] for unit cost, the XP-algorithm and the fixed-parameter approximation scheme for arbitrary cost by Brederick et al. [7].

Further, we have the following:

► **Corollary 4.** *Approval-BRIBERY, Approval-CCAV and Approval-CCDV can be solved in time  $2^{O(|C|^6 \log |C|)} \cdot n^4$ .*

This improves a recent result by Brederick et al. [9] who solved these problems in time that is double-exponential in  $|C|$ .

**Our approach.** Our approach to prove Theorem 1 is to formulate the  $\mathcal{R}$ -MULTI-BRIBERY problem in terms of an  $n$ -fold integer program (IP). Unlike fixed-dimension ILPs, which can be handled by Lenstra’s algorithm [23],  $n$ -fold IPs allow variable dimension at the expense of a more rigid block structure of the constraint matrix. We manage to encode many voting rules  $\mathcal{R}$  in a constraint matrix that has this required structure. While the dimension of the IP is not bounded in terms of the number of candidates, we bound the dimension of each block by a function of  $|C|$ . Then we solve the  $n$ -fold IP via the fixed-parameter algorithm of Hemmecke, Onn and Romanchuk [22], parameterized by the largest coefficient and the largest dimension of each block of the IP.

We complement our positive results by a complexity lower bound for solving  $n$ -fold IPs:

► **Theorem 5.** *Assuming ETH, there is no algorithm solving  $n$ -fold IPs in time  $a^{o(\sqrt[3]{r \cdot s \cdot t})} \cdot n^{O(1)}$ , where  $a$  is the largest absolute value in the constraint matrix and  $r, s, t$  bound the dimension of each block. Further, solving  $n$ -fold IPs is W[1]-hard parameterized by  $r, s, t$ .*

We defer the proof of Theorem 5 to the full version of this paper.

**Related work.** Bribery problems in voting systems are well-studied [7, 13, 14, 19]. Brederick et al. [7] consider shift bribery, where candidates can be shifted up a number of positions in a voter’s preference order; this is a special case of swap bribery. An extension of their model [11] allows campaign managers to affect the position of the preferred candidate in multiple votes, either positively or negatively, with a single bribery action, which applies to large-scale campaigns. In a different model [10], complexity of bribery of elections admitting for multiple winners, such as when committees are formed, has been studied. Also, different cost models have been considered: Faliszewski et al. [18] require that each voter has their own price that is independent of the changes made to the bribed vote. The more general models of Faliszewski [17] and Faliszewski et al. [20] allow for prices that depend on the amount of change the voter is asked for by the briber. For various other bribery models that have been investigated algorithmically, cf. Rothe [3, Chapter 4.3.5].

Regarding ILPs, tractable fragments include ILPs whose defining matrix is totally unimodular (due to the integrality of the corresponding polyhedra and the polynomiality of linear programming), and ILPs in fixed dimension [23]. Courcelle’s theorem [12] implies that solving ILPs is fixed-parameter tractable parameterized by the treewidth of the constraint matrix and the maximum domain size of the variables. Ganian and Ordyniak [21] showed fixed-parameter tractability for the combined parameter the treedepth and the largest absolute value in the constraint matrix, and contrasted this with a W[1]-hardness result when treedepth is exchanged for treewidth.

## 2 Voting and Bribing Problems

We give notions for the problems we deal with; for background, cf. Brams and Fishburn [5].

**Elections.** An election  $(C, V)$  consists of a set  $C$  candidates and a set  $V = V_a \cup V_\ell$  of voters, where  $V_a$  are *active* voters and  $V_\ell$  are *latent* voters. Only active voters participate in an election, but through a control action (defined later) latent voters can become active or active voters can become latent. Unless specified otherwise, we assume that  $V = V_a$ . Each voter  $i$  is a linear order  $\succ_i$  over the set  $C$  which we call a *preference order*. For distinct candidates  $a$  and  $b$ , we write  $a \succ_i b$  if voter  $i$  prefers  $a$  over  $b$ . We denote by  $\text{rank}(c, i)$  the position of candidate  $c \in C$  in the order  $\succ_i$ .

**Swaps.** Let  $(C, V)$  be an election and let  $\succ_i \in V$  be a voter. A *swap*  $\gamma = (a, b)_i$  in preference order  $\succ_i$  means to exchange the positions of  $a$  and  $b$  in  $\succ_i$ ; denote the resulting order by  $\succ_i^\gamma$ ; the *cost* of  $(a, b)_i$  is  $\pi_i(a, b)$ . A swap  $\gamma = (a, b)_i$  is *admissible in  $\succ_i$*  if  $\text{rank}(a, i) = \text{rank}(b, i) - 1$ . A set  $\Gamma$  of swaps is *admissible in  $\succ_i$*  if they can be applied sequentially in  $\succ_i$ , one after the other, in some order, such that each one of them is admissible. Note that the obtained vote, denoted by  $\succ_i^\Gamma$ , is independent from the order in which the swaps of  $\Gamma$  are applied. We also extend this notation for applying swaps in several votes and denote it  $V^\Gamma$ .

**Support changes.** In voting rules such as SP-AV or Fallback, each voter  $\succ_i$  also has an *approval count*  $l_i \in \{0, \dots, |C|\}$ . For voter  $\succ_i$  and  $t \in \{-l_i, \dots, |C| - l_i\}$ , a *support change*  $t_i$  changes the approval count of voter  $i$  to  $l_i + t$ . We denote a set of support changes by  $\Sigma$ , and the changed set of voters by  $V^\Sigma$ . The cost of support change  $t_i$  is  $\alpha_i(t)$ ; always  $\alpha_i(0) = 0$ . If voter  $i$  is involved in a swap or support change, a one-time *influence cost*  $\iota_i$  occurs.

**Control changes.** The set of voters may be changed by activating some latent voters from  $V_\ell$  or deactivating some active voters from  $V_a$ ; we call this a *control change*. We denote the changed set of voters by  $\bar{V}_\ell \cup \bar{V}_a$ . The cost of activating voter  $\succ_i \in V_\ell$  is  $a_i$  and the cost of deactivating voter  $\succ_i \in V_a$  is  $d_i$ ; always  $a_i = 0$  for  $\succ_i \in V_a$  and  $d_i = 0$  for  $\succ_i \in V_\ell$ .

**Voting rules.** A voting rule  $\mathcal{R}$  is a function that maps an election  $(C, V)$  to a subset  $W \subseteq C$ , called the *winners*. We study the following voting rules:

*Scoring protocols.* A scoring protocol is defined through a vector  $\mathbf{s} = (s_1, \dots, s_{|C|}) \in \mathbb{N}_0^{|C|}$  with  $s_1 \geq \dots \geq s_{|C|} \geq 0$ . A candidate receives  $s_j$  points for each voter that ranks it as  $j$ -th best. The candidate with the maximum number of points is the winner. Examples include the Plurality rule ( $\mathbf{s} = (1, 0, \dots, 0)$ ),  $d$ -Approval ( $\mathbf{s} = (1, \dots, 1, 0, \dots, 0)$  with  $d$  ones), and the Borda rule ( $\mathbf{s} = (|C| - 1, |C| - 2, \dots, 1, 0)$ ). Throughout, we assume that  $\max_{j=1}^{|C|} s_j \leq |C|$ , which is the case for the aforementioned popular rules.

*Bucklin.* The *Bucklin winning round* is a number  $k$  such that using the  $k$ -approval rule yields a candidate with more than  $\frac{n}{2}$  points, but  $(k - 1)$ -approval does not. The *Bucklin winner* is the candidate with maximum points when  $k$ -approval is used.

*Condorcet consistent rules.* A candidate  $c \in C$  is a *Condorcet winner* if any other  $c' \in C \setminus \{c\}$  satisfies  $|\{\succ_i \in V \mid c \succ_i c'\}| > |\{\succ_i \in V \mid c' \succ_i c\}|$ . A voting rule is *Condorcet consistent* if it selects the Condorcet winner in case there is one. Such rules are classified by Brams and Fishburn [5] as C1, C2 or C3, depending on the kind of information needed to determine the winner. For  $a, b \in C$  let  $v(a, b) = |\{\succ_i \in V \mid a \succ_i b\}|$ ; we write  $a <_M b$  if  $a$  beats  $b$  in a head-to-head contest, that is, if  $v(a, b) > v(b, a)$ .

Let  $\mathcal{R}$  be a Condorcet consistent rule. We say  $\mathcal{R}$  is C1 if knowing  $<_M$  suffices to determine the winner. C1 rules include the Copeland $^\alpha$  rule, the Slater rule, and others. E.g., the Copeland $^\alpha$  rule for  $\alpha \in [0, 1]$  specifies that for each head-to-head contest between two distinct candidates, if some candidate is preferred by a majority of voters then (s)he obtains one point and the other candidate obtains zero points, and if a tie occurs then both candidates obtain  $\alpha$  points; the candidate with largest sum of points wins.

We say  $\mathcal{R}$  is C2 if it is not C1 and knowing  $v(a, b)$  for all  $a, b \in C$  is sufficient for determining the winner. The following two rules are C2:

*Maximin.* Declares  $c \in C$  is a winner if it maximizes  $v_*(c) = \min\{v(c, a) \mid c \neq a \in C\}$ .

*Kemeny.* Declares  $c \in C$  a winner if there exists a ranking of candidates  $\succ_R$  such that  $c$  is first in  $\succ_R$  and  $\succ_R$  maximizes the total agreement with voters  $\sum_{i=1}^n |\{(a, b) \mid ((a \succ_R b) \Leftrightarrow (a \succ_i b)) \forall a, b \in C\}|$  among all rankings.

We say  $\mathcal{R}$  is C3 if it is neither C2 nor C3. The following two rules are C3:

*Dodgson.* The *Dodgson score* of a candidate  $c$  is the minimum number of swaps needed such that  $c$  becomes the Condorcet winner. A candidate  $c$  is the *Dodgson winner* if their Dodgson score is minimum.

*Young.* Analogously, the *Young score* of a candidate  $c$  is the minimum number of voters that need to be deleted from an election for  $c$  to become the Condorcet winner. The candidate with the lowest Young score is the *Young winner*.

Additionally, if approval counts are given for each voter, other voting rules are possible:

*SP-AV.* A candidate  $c$  gets a point from every voter  $i$  with  $\text{rank}(c, i) \leq l_i$ . The candidate with maximum number of points wins.

*Fallback.* Delete, for each voter  $\succ_i$ , the non-approved candidates (i.e., all  $c$  with  $\text{rank}(c, i) > l_i$ ) from its order. Then, use the Bucklin rule, which might fail due to step one; in that case, use the SP-AV rule.

### 3 A grammar for $n$ -fold integer programming

We now set up our main tool, a grammar for  $n$ -fold IPs. For background on  $n$ -fold IPs, we refer to the books of Onn [25] and De Loera et al. [24].

**$n$ -fold integer programs.** Given  $nt$ -dimensional integer vectors  $\mathbf{b}, \mathbf{u}, \mathbf{l}, \mathbf{w}$ , an  $n$ -fold integer programming problem  $(IP)_{E^{(n)}, \mathbf{b}, \mathbf{l}, \mathbf{u}, \mathbf{w}}$  in variable dimension  $nt$  is defined as

$$\min \left\{ \mathbf{w}\mathbf{x} : E^{(n)}\mathbf{x} = \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \in \mathbb{Z}^{nt} \right\}, \quad \text{where } E^{(n)} := \begin{pmatrix} D & D & \cdots & D \\ A & 0 & \cdots & 0 \\ 0 & A & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A \end{pmatrix}$$

is an  $(r + ns) \times nt$ -matrix,  $D \in \mathbb{Z}^{r \times t}$  is an  $r \times t$ -matrix and  $A \in \mathbb{Z}^{s \times t}$  is an  $s \times t$ -matrix.

Hemmecke et al. [22] developed a dynamic program to show:

► **Proposition 6** ([22, Thm. 6.1]). *There is an algorithm solving  $(IP)_{E^{(n)}, \mathbf{b}, \mathbf{l}, \mathbf{u}, \mathbf{w}}$  in time  $a^{O(trs+t^2s)} \cdot O(n^3L)$ , where  $a = \max\{\|D\|_\infty, \|A\|_\infty\}$  and  $L$  is the length of the input.*

The structure of  $E^{(n)}$  allows us to divide the  $nt$  variables into  $n$  bricks of size  $t$ . We use subscripts to index within a brick and superscripts to denote the index of the brick, i.e.  $x_j^i$  is the  $j$ -th variable of the  $i$ -th brick with  $j \in \{1, \dots, t\}$  and  $i \in \{1, \dots, n\}$ .

Also note that there are two types of constraints, given by either the row of matrices  $D$ , or by the matrix  $A$  for each brick. These types lead us to define globally and locally uniform expressions. A *globally uniform expression* for a  $t$ -tuple of coefficients  $(a_1, \dots, a_t)$  is:

$$\sum_{i=1}^n \sum_{j=1}^t a_j x_j^i .$$

Observe that constraints given by a row of matrices  $D$  have the form of a globally uniform expression equalling a number on the right hand side. We call them *globally uniform constraints*; typically they assure that the solution fits a budget or other global condition.

A *locally uniform expression* for  $(a_1, \dots, a_t)$  is an expression

$$\sum_{j=1}^t a_j x_j^i - b_i, \quad i = 1, \dots, n .$$

Notice that a locally uniform expression is in fact a set of  $n$  expressions which only differ in their additive constants. All constraints which are not globally uniform have the form of a locally uniform expression equalling some right hand side. We call them *locally uniform constraints* and typically use them to give variables within a brick their intended meaning.

As we have just seen, unlike with general IPs,  $n$ -fold IPs obey a uniform block structure. Many “integer programming tricks” are known for expressing logical connectives and other operations within IP; however, it is not obvious if they can be implemented also in  $n$ -fold IP. Our goal is to establish that all constraints constructed in a particular way are valid uniform constraints, and determine the parameters  $r, s, t$  and  $a$  in the run time of Theorem 6.

**Expressions and constraints.** We define an  *$n$ -fold IP grammar* as follows, where **lue** and **gue** stands for “locally uniform expression” and “globally uniform expression”:

$$\begin{aligned} \heartsuit &::= = | < | > | \leq | \geq \\ \diamond &::= \heartsuit \neq \\ \mathbf{lue}_m &::= x^i \quad | \quad \text{a variable per brick with } l^i, u^i \text{ s.t. } \max_i u^i - \min_i l^i \leq m \\ &\quad \sum_{j=1}^k a_j \mathbf{lue}_{m_j} \quad | \quad \text{for } k \text{ integers } a_1, \dots, a_k \\ &\quad \mathbf{lue}_m + o^i \quad | \quad \text{for } n \text{ integers } o^1, \dots, o^n \\ &\quad (\mathbf{lue}_m) \quad | \quad \text{to clarify operator priority} \\ &\quad \lambda \quad | \quad \text{an empty expression} \\ \mathbf{gue} &::= \mathbf{lue}_m \end{aligned}$$

The subscript  $\mathbf{lue}_m$  denotes an external guarantee that the result of this expression lies in  $\{L, \dots, U\}$  with  $|U - L| \leq m$ . Notice that in the above, one **lue** actually describes  $n$  objects, one in each brick. These objects differ from each other in exactly two ways. One, when a variable  $x^i$  is added to each brick, its lower and upper bounds  $l^i, u^i$  may be different for each  $i \in \{1, \dots, n\}$ . Second, in the expression  $\mathbf{lue}_m + o^i$ , the additive constant  $o^i$  may be different for each  $i \in \{1, \dots, n\}$ .

## 46:8 Voting and Bribing in Single-Exponential Time

Special attention is given to binary  $\mathbf{lue}$ :  $\mathbf{lue}_b$  is  $\mathbf{lue}_1$  which is always in  $\{0, 1\}$ .

$$\begin{aligned} \mathbf{lue}_b &::= \mathbf{lue}_b \rightarrow \mathbf{lue}_b \mid \neg \mathbf{lue}_b \mid \mathbf{lue}_b \wedge \mathbf{lue}_b \mid \\ &\quad \mathbf{lue}_b \vee \mathbf{lue}_b \mid \mathbf{lue}_b \oplus \mathbf{lue}_b \mid \\ &\quad \text{bool}_m(\mathbf{lue}_m) \mid && 0 \text{ if } \mathbf{lue}_m \text{ is } 0, \text{ else } 1 \\ &\quad \text{bool}_m(\mathbf{lue}_m \diamond \mathbf{lue}_m) && 1 \text{ if } \mathbf{lue}_m \diamond \mathbf{lue}_m, \text{ else } 0 \\ \mathbf{lue}_2 &::= \text{sign}_m(\mathbf{lue}_m) && -1 \text{ if } \mathbf{lue}_m \text{ neg.}, 1 \text{ if pos.}, 0 \text{ otherwise.} \end{aligned}$$

Then, locally and globally uniform constraints ( $\mathbf{luc}$  and  $\mathbf{guc}$ ) are defined as:

$$\mathbf{luc} ::= \mathbf{lue}_m \diamond \mathbf{lue}_{m'} \quad \text{and} \quad \mathbf{guc} ::= \mathbf{gue} \heartsuit \mathbf{gue}$$

Finally, an  $n$ -fold IP recipe is a tuple  $(\mathcal{G}, \mathcal{L})$  where  $\mathcal{G}$  and  $\mathcal{L}$  are the sets of locally and globally uniform constraints, respectively. Let  $t' \in \mathbb{N}$  be the number of variables introduced to each brick. Let  $\text{Sol}(\mathcal{G}, \mathcal{L}) \subseteq \mathbb{Z}^{nt'}$  be the set of all vectors that satisfy all constraints in  $(\mathcal{G} \cup \mathcal{L})$ .

**Constructing  $n$ -fold IP from recipe.** Now we shall describe how to turn an  $n$ -fold IP recipe into a system of lower and upper bounds and linear equalities satisfying the  $n$ -fold format, proving this theorem:

► **Theorem 7.** *Let  $(\mathcal{G}, \mathcal{L})$  be an  $n$ -fold IP recipe with  $\mathcal{G} \cup \mathcal{L}$  build by  $m$  applications of the grammar rules such that  $a$  is an upper bound on  $|a_j|$  in all coefficients and all  $m'$  appearing in  $\text{bool}_{m'}$  and  $\text{sign}_{m'}$ . Then in time  $O(|\mathcal{G} \cup \mathcal{L}|)$ , one can compute numbers  $r, s, t \in O(m)$ , an  $n$ -fold matrix  $E^{(n)} \in \mathbb{Z}^{nt}$ , a right hand side  $\mathbf{b} \in \mathbb{Z}^{r+sn}$  and lower and upper bounds  $\mathbf{l}, \mathbf{u} \in \mathbb{Z}^{r+sn}$  such that  $O(a)$  is an upper bound on the absolute value of the coefficients of  $E^{(n)}$ , and  $\text{Sol}(\mathcal{G}, \mathcal{L}) = \{(\mathbf{x}^1|_{t'}, \dots, \mathbf{x}^n|_{t'} \mid E^{(n)}(\mathbf{x}^1, \dots, \mathbf{x}^n) = \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\}$ , where  $\mathbf{x}^i|_{t'}$  is the restriction of  $\mathbf{x}^i$  to its first  $t'$  variables.*

For the proof, to determine the parameters  $r, s, t$  of the resulting  $n$ -fold IP, we define the  $s$ -increase of  $\mathbf{lue}_m$  and  $t$ -increase of  $\mathbf{lue}_m$ , denoted  $\Delta s(\mathbf{lue}_m)$  and  $\Delta t(\mathbf{lue}_m)$ , and analogously for  $\mathbf{luc}$ 's and  $\mathbf{gue}$ 's. The  $s$ -increase  $\Delta s(\mathbf{lue}_m)$  and the  $t$ -increase  $\Delta t(\mathbf{lue}_m)$  are the number of auxiliary equalities and auxiliary variables needed to express a  $\mathbf{lue}_m$  or  $\mathbf{luc}$ . We note the  $s$ - and  $t$ -increase of each rule after defining it.

**Rewriting  $\mathbf{lue}_m$  to  $\sum_{j=1}^k a_j x_j + \bar{b}^i$  for  $i = 1, \dots, n$ .** Rewriting  $\mathbf{lue}_m$  in some expression means replacing it with a new variable  $z$  and adding locally uniform constraints to  $\mathcal{L}$ . These constraints assure that the variable  $z$  will carry the desired meaning. The result is that every  $\mathbf{lue}_m$  is rewritten to the format  $\sum_{j=1}^t a_j x_j + b^i$  as introduced in Sect. 3. In this phase we may still be adding constraints that are not in the  $n$ -fold format (contain inequalities etc.) as they will be dealt with later. Also note that we use the notation  $\overline{\mathbf{lue}}$  simply to distinguish between various  $\mathbf{lue}$ ; do not confuse this with negation.

- $\mathbf{lue}_b ::= \mathbf{lue}'_b \vee \overline{\mathbf{lue}_b} \Rightarrow$  create a new binary variable  $s$  and set  $2z = \mathbf{lue}'_b + \overline{\mathbf{lue}_b} + s$ .  
 $\Delta s(\mathbf{lue}_b) = \Delta s(\mathbf{lue}'_b) + \Delta s(\overline{\mathbf{lue}_b}) + 1$ ,  $\Delta t(\mathbf{lue}_b) = \Delta t(\mathbf{lue}'_b) + \Delta t(\overline{\mathbf{lue}_b}) + 2$ .
- $\mathbf{lue}_b ::= \neg \mathbf{lue}'_b \Rightarrow z = 1 - \mathbf{lue}'_b$ .  
 $\Delta s(\mathbf{lue}_b) = \Delta s(\mathbf{lue}'_b) + 1$ ,  $\Delta t(\mathbf{lue}_b) = \Delta t(\mathbf{lue}'_b)$ .
- $\mathbf{lue}_b ::= \mathbf{lue}_b \rightarrow \mathbf{lue}_b \mid \mathbf{lue}_b \wedge \mathbf{lue}_b \mid \mathbf{lue}_b \oplus \mathbf{lue}_b$ : it is folklore [16] that all logical connectives can be constructed with  $\neg$  and  $\vee$ .  
 $\Delta s(\mathbf{lue}_b) = \Delta s(\mathbf{lue}'_b) + \Delta s(\overline{\mathbf{lue}_b}) + O(1)$ ,  $\Delta t(\mathbf{lue}_b) = \Delta t(\mathbf{lue}'_b) + \Delta t(\overline{\mathbf{lue}_b}) + O(1)$

- $\mathbf{lue}_b ::= \text{bool}_m(\mathbf{lue}_m)$  and  $\mathbf{lue}_2 ::= \text{sign}_m(\mathbf{lue}_m)$ : We assume that  $-L < \mathbf{lue}_m < U$  for  $L, U \in \mathbb{N}$  and that  $m = \max\{L, U\}$ <sup>2</sup>. Because we will introduce a coefficient upper bounded by  $m$  into the system, we will denote the operation  $\text{bool}_m(x)$ . Let  $v, u$  be two new variables such that  $v = 1$  if and only if  $\mathbf{lue}_m \geq 0$ , and  $u = 1$  if and only if  $\mathbf{lue}_m \leq 0$ :

$$v, u \in \{0, 1\}: \quad 1 + \mathbf{lue}_m \leq Uv \leq U + \mathbf{lue}_m \quad \& \quad 1 - \mathbf{lue}_m \leq Lu \leq L - \mathbf{lue}_m$$

Then if  $\mathbf{lue}_b ::= \text{bool}_m(\mathbf{lue}_m)$  let  $z = \neg(v \wedge u)$  and if  $\mathbf{lue}_b ::= \text{sign}_m(\mathbf{lue}_m)$  let  $z = v - u$ .  $\Delta s(\mathbf{lue}_b) = \Delta(\mathbf{lue}_m) + O(1)$ ,  $\Delta t(\mathbf{lue}_b) = \Delta t(\mathbf{lue}_m) + O(1)$  and analogously for  $\mathbf{lue}_2$ .

- $\mathbf{lue}_b ::= \text{bool}_m(\mathbf{lue}'_m \diamond \overline{\mathbf{lue}_m}) \Rightarrow$ 
  - $\diamond$  is “=”:  $z = \text{bool}_m(\mathbf{lue}'_m - \overline{\mathbf{lue}_m})$
  - $\diamond$  is “ $\neq$ ”:  $z = 1 - \text{bool}_m(\mathbf{lue}'_m - \overline{\mathbf{lue}_m})$
  - $\diamond$  is “ $>$ ”:  $z = \text{bool}_m(\text{sign}_m(\mathbf{lue}'_m - \overline{\mathbf{lue}_m}) = 1)$
  - $\diamond$  is “ $\geq$ ”:  $z = \text{bool}_m(\mathbf{lue}'_m > \overline{\mathbf{lue}_m}) \vee \text{bool}_m(\mathbf{lue}'_m = \overline{\mathbf{lue}_m})$

And analogously when  $\diamond$  is “ $<$ ” and “ $\leq$ ”.

$\Delta s(\mathbf{lue}_b) = \Delta s(\mathbf{lue}'_b) + \Delta s(\overline{\mathbf{lue}_b}) + O(1)$ ,  $\Delta t(\mathbf{lue}_b) = \Delta t(\mathbf{lue}'_b) + \Delta t(\overline{\mathbf{lue}_b}) + O(1)$
- $\mathbf{lue}_m ::= \lambda \mid (\mathbf{lue}_m) \mid \mathbf{lue}_m + o^i \mid \sum_{j=1}^k a_j \mathbf{lue}_{m_j} \mid x^i$  are rewritten in the obvious way.

**Rewriting  $\mathbf{luc} ::= \mathbf{lue}_m \diamond \mathbf{lue}_{m'}$  to  $\mathbf{lue}_{m''} = \mathbf{b}^i$**

- when  $\diamond$  is “=”:  $\mathbf{lue}_m = \mathbf{lue}_{m'} \Rightarrow \mathbf{lue}_m - \mathbf{lue}_{m'} = 0$ .  
 $\Delta s(\mathbf{luc}) = \Delta s(\mathbf{lue}_m) + \Delta s(\mathbf{lue}_{m'})$  and  $\Delta t(\mathbf{luc}) = \Delta t(\mathbf{lue}_m) + \Delta t(\mathbf{lue}_{m'})$ .
- when  $\diamond$  is “ $\neq$ ”:  $\mathbf{lue}_m \neq \mathbf{lue}_{m'} \Rightarrow \text{bool}_{m+m'}(\mathbf{lue}_m \neq \mathbf{lue}_{m'}) = 1$ . Then,  
 $\Delta s(\mathbf{luc}) = \Delta s(\text{bool}_{m+m'}(\mathbf{lue}_m \neq \mathbf{lue}_{m'}))$  and  $\Delta t(\mathbf{luc}) = \Delta t(\text{bool}_{m+m'}(\mathbf{lue}_m \neq \mathbf{lue}_{m'}))$ .
- when  $\diamond$  is not “=”, intuitively we want to add a slack variable:  
 $\mathbf{lue}_m \diamond \mathbf{lue}_{m'} \Rightarrow \mathbf{lue}_m - \mathbf{lue}_{m'} + \sum_{i=1}^n y^i = 0$  with  $y^i$  for  $i = 1, \dots, n$  being  $n$  new variables with  $l^i = u^i = 0$  for  $i > 1$  and with
  - $l^1 = 0$  and  $u^1 = \infty$  when  $\diamond$  is “ $\leq$ ”,
  - $l^1 = 1$  and  $u^1 = \infty$  when  $\diamond$  is “ $<$ ”,
  - $l^1 = -\infty$  and  $u^1 = 0$  when  $\diamond$  is “ $\geq$ ”, and
  - $l^1 = -\infty$  and  $u^1 = -1$  when  $\diamond$  is “ $>$ ”,

“ $\infty$ ” stands for a sufficiently large number, which is usually clear from the context. Then,  
 $\Delta s(\mathbf{luc}) = \Delta s(\mathbf{lue}_m) + \Delta s(\mathbf{lue}_{m'})$  and  $\Delta t(\mathbf{luc}) = \Delta t(\mathbf{lue}_m) + \Delta t(\mathbf{lue}_{m'}) + 1$ .

Finally, rewrite the globally uniform constraints in  $\mathcal{G}$ . Since  $\mathbf{gue} ::= \mathbf{lue}_m$  and  $\mathbf{lue}_m$  gets rewritten to  $\sum_{j=1}^t a_j x_j^i$  as required, the above rules suffice to obtain a globally uniform expression as defined in Sect. 3, and similarly for  $\mathbf{guc} ::= \mathbf{gue} \heartsuit \mathbf{gue}'$ . Let  $\Delta t(\mathbf{guc}) = \Delta t(\mathbf{gue}) + \Delta t(\mathbf{gue}')$  if  $\heartsuit$  is “=” and  $\Delta t(\mathbf{guc}) = \Delta t(\mathbf{gue}) + \Delta t(\mathbf{gue}') + 1$  otherwise and let  $\Delta t(\mathbf{gue}) = \Delta t(\mathbf{lue}_m)$ .

It remains to compute the parameters  $r, s, t$ . Let  $\#\text{variables}$  be the number of distinct variables introduced through  $\mathbf{lue}_m ::= x^i$ . Then,

- $t = \#\text{variables} + \sum_{\mathbf{luc}_j \in \mathcal{L}} \Delta t(\mathbf{luc}_j) + \sum_{\mathbf{guc}_j \in \mathcal{G}} \Delta t(\mathbf{guc}_j)$ ,
- $s = |\mathcal{L}| + \sum_{\mathbf{luc}_j \in \mathcal{L}} \Delta s(\mathbf{luc}_j)$ ,
- $r = |\mathcal{G}|$ , and
- all coefficients are bounded in absolute value by  $\max\{\max\{|a_j| \mid a_j \text{ in } \sum_{j=1}^k a_j \mathbf{lue}_m\}, \max\{m \mid \text{bool}_m, \text{sign}_m \in \mathbf{luc} \in \mathcal{L}\}\} \in O(a)$ .

Clearly,  $r, s, t \in O(m)$  and this concludes the proof of Theorem 7.

<sup>2</sup> This is without loss of generality: when  $L < \mathbf{lue}_m < U$  with  $L, U \in \mathbb{N}$ ,  $\mathbf{lue}_m$  is always positive and  $\text{bool}_m(\mathbf{lue}_m)$  is always 1; analogously when  $\mathbf{lue}_m$  is always negative.

**A demonstration of the rewriting process.** We want  $m$  variables  $x_1, \dots, x_m$  in each brick describing a permutation of  $\{1, \dots, m\}$ . That is equivalent to  $\sum_{j=1}^m x_j = \binom{m+1}{2}$  and  $x_j \neq x_k$  for all  $j \neq k$  and  $x_j \in \{1, \dots, m\}$  for all  $j$ . This is expressible by  $1 + \binom{m}{2}$  `luc`'s:

$$\sum_{j=1}^m x_j = \binom{m+1}{2} \quad \bigwedge \quad x_j \neq x_k \quad \text{for all } j \neq k . \quad (1)$$

The first `luc` is already in the format required in Sect. 3. However, for the other `luc`'s, rewriting rules are applied. Fix  $j, k$ . The resulting  $n$ -fold IP will contain these constraints (for brevity we omit rewriting inequalities by slack variables as this is standard):

$$\begin{aligned} w &= x_j - x_k \\ 1 + w &\leq mv \leq m + w \\ 1 - w &\leq mu \leq m - w && \text{express } z_{\neg \text{bool}} = \neg(v \wedge u) = \neg v \vee \neg u \\ v_{\neg} &= 1 - v \quad \bigwedge \quad u_{\neg} = 1 - u && v, u, s \in \{0, 1\} \\ 2z_{\vee} &= v_{\neg} + u_{\neg} + s \\ z_{\neg \text{bool}} &= 1 - z_{\vee} && \text{and set } \neg \text{bool}(x_j - x_k) = 0 \\ z_{\neg \text{bool}} &= 0 . \end{aligned}$$

Further, given  $n$  permutations  $o_1^i, \dots, o_m^i$  for  $i = 1, \dots, n$ , one for each brick, we want to compare the permutation  $x_1, \dots, x_m$  to  $o_1, \dots, o_m$  and determine which indices are inverted, that is,  $x_j < x_k \Leftrightarrow o_j > o_k$ . In other words, we want to determine when the sign of  $(x_j - x_k)$  equals the sign of  $(o_k - o_j)$ . So, for each  $j \neq k$  we add a new indicator variable  $s_{jk}$  as follows:

$$s_{jk} = \text{bool}_2(\text{sign}_m(x_j - x_k) = \text{sign}_m(o_k^i - o_j^i)) \quad (2)$$

Notice that  $\text{sign}_m(o_k^i - o_j^i)$  is a constant  $o_{kj}^i$ , so the expression above turns to  $\text{bool}_2(\text{sign}_m(x_j - x_k) - o_{kj}^i)$  which is now clearly a `lueb` and is rewritten similarly as before.

An example of what is *not* expressible in the  $n$ -fold IP grammar is the (nonsensical) expression  $\sum_{j=1}^m o_j^i x_j$ . This is not a `lue` since the numbers  $o_j^i$  appear not as additive constants but as coefficients. However, coefficients are required to be identical across bricks in the grammar rule  $\text{lue}_m ::= \sum_{j=1}^k a_j \text{lue}_{m_j}$ .

► **Remark.** Naturally, we ask if the `bool()` operation can be implemented *without* introducing a number  $a$  depending on the lower and upper bounds, as  $a$  becomes the base of the run time in Theorem 6. One can show that such dependence is necessary (proof deferred):

► **Lemma 8.** *Unless  $\text{FPT} = \text{W}[1]$ , for any computable function  $f$  it is impossible to express the `bool()` operation consistently with the  $n$ -fold IP format while introducing only numbers bounded by  $f(k)$ , and introducing only  $f(k)$  new variables. Moreover, solving  $n$ -fold IP parameterized only by the dimensions  $r, s, t$  (and not by the largest entry  $a$ ) is  $\text{W}[1]$ -hard.*

## 4 Single-Exponential Algorithms for Voting and Bribing

We now establish a formulation of  $\mathcal{R}$ -MULTI-BRIBERY as an  $n$ -fold IP, for various rules  $\mathcal{R}$ . To this end, we first describe the part of the IP which is common to all such rules. Thereafter, we add the parts of the formulation which depend on  $\mathcal{R}$ . Given an instance  $(C, V)$  of  $\mathcal{R}$ -MULTI-BRIBERY, we construct an  $n$ -fold IP whose variables describe the situation after bribery actions (swaps, support changes, control changes) were performed. From these variables we also derive new variables to express the cost function. In the following we always describe the variables and constraints added per voter, and there is one brick per voter.



**Swaps.** We describe the preference order with swaps  $\Gamma$  applied by variables  $x_1, \dots, x_{|C|}$  with the intended meaning  $x_j^i = \text{rank}(c_j, i)^\Gamma$ . Recall from Sect. 3 that constraints (1) enforce that  $(x_1, \dots, x_{|C|})$  is a permutation of  $\{1, \dots, |C|\}$ ; we add them to the program.

To express the swaps performed by  $\Gamma$ , for each pair of candidates  $c_j, c_k \in C$  we introduce binary variables  $s_{jk}, s_{kj}$  so that  $s_{jk} = 1$  if and only if  $c_j$  and  $c_k$  are swapped. We use the fact (cf. [15, Proposition 3.2]) that for two orders  $\succ, \succ'$ , the admissible set of swaps  $\Gamma$  such that  $\succ' = \succ^\Gamma$  is uniquely given as the set of pairs  $(c_i, c_j)$  for which either  $c_i \succ c_j \wedge c_j \succ' c_i$  or  $c_j \succ c_i \wedge c_i \succ' c_j$ . Thus, we only need to set constraint (2) from Sec. 3 with  $\sigma_j^i = \text{rank}(c_j, i)$ .

**Support changes.** To indicate support changes, we introduce binary variables  $r_{-|C|}, \dots, r_{|C|}$  where  $r_0 = 1$  means no change,  $r_j = 1$  means support change  $j$ . We set the lower and upper bounds to ensure that  $r_j = 0$  for all  $j \notin \{-l_i, |C| - l_i\}$ . Finally, we introduce a variable  $x_\alpha \in \{1, \dots, |C|\}$  indicating the approval count after the support change:

$$\sum_{j=-|C|}^{|C|} r_j = 1, \quad x_\alpha = l_i + \sum_{j=-|C|}^{|C|} j r_j .$$

**Influence bit.** We introduce a binary variable  $x_\iota$  taking value 1 if a swap or a support change is performed, and value 0 otherwise:  $x_\iota = \text{bool}_{|C|^2+2|C|} \left( \sum_{j \neq k} s_{jk} + \sum_{j \neq 0} r_j \right)$ .

**Control changes.** We introduce two binary variables  $x_a, x_\ell$  such that  $x_a = 1, x_\ell = 0$  if voter  $i$  is active, and  $x_a = 0, x_\ell = 1$  if voter  $i$  latent:  $x_a + x_\ell = 1$ .

Further, for each pair  $c_j, c_k \in C$  of candidates we introduce a variable  $x_{jk}$  which takes value 1 if  $c_j \succ_i^? c_k$  and 0 otherwise. We will also frequently (and implicitly) use the following variable-splitting trick:

► **Lemma 9.** *Let  $x$  be an integral variable with lower bound  $\ell$  and upper bound  $u$  and let  $z$  be a binary variable. It is possible to introduce a variable  $x^z$  and auxiliary **lue** constraints with  $\Delta s = O(1)$  and  $\Delta t = O(1)$  such that  $x^z = x$  if  $z = 1$  and  $x^z = 0$  if  $z = 0$ .*

**Proof.** First, add  $\ell z \leq x^z \leq uz$ ; then  $z = 0$  implies  $x^z = 0$  and does not influence  $x^z$  when  $z = 1$ . Second, add  $\text{bool}_b(\text{bool}_b(z = 1) \rightarrow \text{bool}_m(x^z = x) = 1)$  with  $m = O(-\ell + u)$ . ◀

**Objective function.** Finally, the linear objective function is given as follows:

$$w(\mathbf{x}, \mathbf{s}, \mathbf{r}) = \sum_{i=1}^n \left[ \left( \sum_{j \neq k} \pi_i(j, k) s_{jk}^i \right) + \left( \sum_{j=-m}^m \alpha_i(j) r_j^i \right) + \iota_i x_\iota^i + a_i x_a^i + d_i x_\ell^i \right] .$$

Let us determine the values of the parameters  $r, s, t$  and  $a$ . We have introduced  $O(|C|^2)$  variables and imposed  $O(|C|^2)$  constraints on them, so  $s = t = O(|C|^2)$ . The coefficients  $a_j$  obey  $\max_j |a_j| \leq O(|C|)$ , and we use the  $\text{bool}_M$  operation with  $M \in O(|C|^2)$ , so  $a = O(|C|^2)$ .

Now we describe the part specific to the voting rules. A voting rule  $\mathcal{R}$  is incorporated in the IP in two steps. First, optionally, new variables are derived using locally uniform constraints. Then, globally uniform constraints are imposed.

Often we can only set up the IP knowing certain facts about how the winning condition is satisfied. We guess those facts, construct the IP, solve it and remember the objective value. Finally, we choose the minimum over all guesses.

**Scoring protocol  $\mathbf{s} = (s_1, \dots, s_{|C|})$ .** We introduce variables  $\sigma_c$  where  $\sigma_c$  is the number of points a voter gives candidate  $c$ . Then, an “active” copy of each new variable (denoted  $\sigma_i^a$  for  $i = 1, \dots, |C|$ ) is created such that we can disregard the contribution of latent voters. To

## 46:12 Voting and Bribing in Single-Exponential Time

do this we use Lemma 9 with  $x := \sigma_i$ ,  $z := x^a$ , and  $x^z := \sigma_i^a$  for every  $i = 1, \dots, |C|$ . Then we add the following globally uniform constraints:

$$\sum_{i=1}^n \sigma_j^a < \sum_{i=1}^n \sigma_1^a \text{ for } j = 2, \dots, |C| .$$

**Any C1 rule  $\mathcal{R}$ .** We guess the resulting  $<_M$  such that  $c_1$  is a winner with respect to  $\mathcal{R}$ ; there are  $O(3^{|C|^2})$  guesses. Knowing  $<_M$  means that for any pair  $c_j, c_k$  of distinct candidates, we know if  $v(c_j, c_k) > v(c_k, c_j)$ ,  $v(c_k, c_j) > v(c_j, c_k)$  or  $v(c_j, c_k) = v(c_k, c_j)$ . We thus add:

$$\sum_{i=1}^n x_{jk}^a > \sum_{i=1}^n x_{kj}^a \text{ (if } c_j <_M c_k) \text{ and } \sum_{i=1}^n x_{jk}^a = \sum_{i=1}^n x_{kj}^a \text{ (if } v(c_j, c_k) = v(c_k, c_j)) .$$

**Maximin rule.** For  $c_1$  to be winner with the maximin rule means that there is a  $B \in \{1, \dots, |V|\}$  such that  $v_*(c_1) = B$ , while for all  $c \in C \setminus \{c_1\}$ ,  $v_*(c) < B$ . That, in turn, means, that for every candidate  $c_j$  there is a candidate  $c_{j'}$  such that  $v(c_j, c_{j'}) < B$ . Guess  $B$  and  $c_{j'}$  for every  $c_j$ ; there are at most  $n \cdot |C|^2$  guesses. Then add the following constraints:

$$\sum_{i=1}^n x_{1j}^a \geq B \quad \text{and} \quad \sum_{i=1}^n x_{jj'}^a < B, \quad j = 2, \dots, |C| .$$

**Bucklin.** Guess the number  $|\bar{V}_a| \in \{1, \dots, n\}$  of active voters and set  $\sum_{i=1}^n x_a^i = |\bar{V}_a|$ . Then, guess the winning round  $k$  and note that the winning score will be larger than  $|\bar{V}_a|/2$ . Altogether there are  $O(|C||V|)$  guesses. Similarly to scoring protocols, we introduce variables  $\sigma_j$  (number of points for candidate  $j$  in  $k$ -approval) and  $\tilde{\sigma}_j$  (number of points for candidate  $j$  in  $(k-1)$ -approval). We omit the details of how to split variables into active and latent:

$$\sigma_j = \text{bool}_{|C|}(x_j < k) \quad \text{and} \quad \tilde{\sigma}_j = \text{bool}_{|C|}(x_j < k-1), \quad j = 1, \dots, |C| .$$

Then, the winning condition is expressed as:

$$\sum_{i=1}^n \sigma_1^a > |\bar{V}_a|/2 \quad \bigwedge \quad \sum_{i=1}^n \sigma_j^a < \sum_{i=1}^n \sigma_1^a \text{ for } j = 2, \dots, |C| \quad \bigwedge \quad \sum_{i=1}^n \tilde{\sigma}_c^a < |\bar{V}_a|/2, c \in C .$$

**SP-AV.** In SP-AV, each candidate  $c$  receives a point if it ranks above the approval count. As before, we introduce variables  $\sigma_c$  for points received by a candidate  $c$  and split them into active and latent (again using Lemma 9).

$$\sigma_j = \text{bool}_{|C|}(x_j < x_\alpha) \text{ for } j = 1, \dots, |C| \quad \bigwedge \quad \sum_{i=1}^n \sigma_j^a < \sum_{i=1}^n \sigma_1^a \text{ for } j = 2, \dots, |C| .$$

**Fallback.** In the fallback rule, the non-approved candidates are discarded, the Bucklin rule is applied and if it fails to select a winner, the SP-AV rule is applied. We guess the Bucklin winning round  $k$  or if SP-AV is used and the number  $|\bar{V}_a|$  of active voters; there are  $O(|V| \cdot |C|)$  guesses. (SP-AV is used exactly when the winning score is less than  $|\bar{V}_a|/2$ .) If Bucklin is used, we need a slight modification to take support changes into account. Instead of  $\sigma_j = \text{bool}_{|C|}(x_j < k)$  we have  $\sigma_j = [\text{bool}_{|C|}(x_j < k) \wedge \text{bool}_{|C|}(k \leq x_\alpha)]$ ; similarly for  $\tilde{\sigma}_j$ .

As argued, for each rule we introduced  $O(|C|^2)$  variables and locally uniform constraints, and  $O(|C|^2)$  globally uniform constraints. Thus, by Theorem 7 we get an  $n$ -fold matrix with parameters  $r = s = t = a = O(|C|^2)$ . Proposition 6 is then used to solve the  $n$ -fold IP in time  $2^{O(|C|^6 \log |C|)} n^3$ . Also,  $O(3^{|C|^2})$  guesses suffice for each rule except Maximin, Bucklin and Fallback, where  $O(|C|^2|V|)$  guesses suffice.

An exception to this run time is the Kemeny rule:

**Kemeny.** For  $c_1$  to be a Kemeny winner, there has to be a ranking  $\succ_{R^*}$  of the candidates that ranks  $c_1$  first and  $\succ_{R^*}$  maximizes the total agreement with voters  $\sum_{i=1}^{|V|} \{(a, b) \mid ((a \succ_{R^*} b) \Leftrightarrow (a \succ_i b)), a, b \in C\}$  among all rankings. In other words, the number of swaps sufficient to transform every  $\succ_i$  into  $\succ_{R^*}$  is smaller than the number of swaps needed to transform every  $\succ_i$  into any other  $\succ_{R'}$  where  $c_1$  is not first.

We guess the ranking  $\succ_{R^*}$ ; then we introduce variables  $x_R^i$  for  $R \in \{R^*\} \cup \{R' \mid c_1 \text{ is not first in } R'\}$  so that  $x_R^i$  is the number of swaps needed to transform  $\succ_i^\Gamma$  into  $\succ_R$ , splitting them into active and latent as before. Then, we introduce the necessary constraints:

$$\begin{aligned} x_R &= \sum_{j \neq k} [\text{sign}_{|C|}(x_j - x_k) = \text{sign}_{|C|}(\text{rank}(k, R) - \text{rank}(j, R))] && \text{for all } R, \\ \sum_{i=1}^n x_R^{a_i} &> \sum_{i=1}^n x_{R^*}^{a_i} && \text{for } R \neq R^* . \end{aligned}$$

This solves Kemeny-MULTI-BRIBERY in time  $2^{O(|C|^6)} n^3$ , and completes proving Theorem 1.

---

## References

- 1 John J. Bartholdi III, Craig A. Tovey, and Michael A. Trick. Voting schemes for which it can be difficult to tell who won the election. *Soc. Choice Welfare*, 6(2):157–165, 1989.
- 2 Dorothea Baumeister, Piotr Faliszewski, Jérôme Lang, and Jörg Rothe. Campaigns for lazy voters: truncated ballots. In *Proc. AAMAS 2012*, pages 577–584, 2012.
- 3 Dorothea Baumeister and Jörg Rothe. Preference aggregation by voting. In Jörg Rothe, editor, *Economics and Computation. An Introduction to Algorithmic Game Theory, Computational Social Choice, and Fair Division.*, pages 197–326. Springer, 2016.
- 4 Nadja Betzler, Susanne Hemmann, and Rolf Niedermeier. A multivariate complexity analysis of determining possible winners given incomplete votes. In *Proc. IJCAI 2009*, pages 53–58, 2009.
- 5 Steven J. Brams and Peter C. Fishburn. Voting procedures. In Katora Suzumura Kenneth J. Arrow, Armatya K. Sen, editor, *Handbook of Social Choice and Welfare*, volume 19 of *Handbooks in Economics*, pages 173–236. Elsevier, 2002.
- 6 Robert Brederbeck, Jiehua Chen, Piotr Faliszewski, Jiong Guo, Rolf Niedermeier, and Gerhard J. Woeginger. Parameterized algorithmics for computational social choice: Nine research challenges. *Tsinghua Sci. Tech.*, 19(4):358–373, 2014.
- 7 Robert Brederbeck, Jiehua Chen, Piotr Faliszewski, André Nichterlein, and Rolf Niedermeier. Prices matter for the parameterized complexity of shift bribery. In *Proc. AAAI 2014*, pages 552–558, 2014.
- 8 Robert Brederbeck, Jiehua Chen, Sepp Hartung, Stefan Kratsch, Rolf Niedermeier, Ondrej Suchý, and Gerhard J. Woeginger. A multivariate complexity analysis of lobbying in multiple referenda. *J. Artificial Intelligence Res.*, 50:409–446, 2014.
- 9 Robert Brederbeck, Piotr Faliszewski, Rolf Niedermeier, Piotr Skowron, and Nimrod Talmon. Elections with few candidates: Prices, weights, and covering problems. In *Proc. ADT 2015*, volume 9346 of *Lecture Notes Comput. Sci.*, pages 414–431, 2015.
- 10 Robert Brederbeck, Piotr Faliszewski, Rolf Niedermeier, Piotr Skowron, and Nimrod Talmon. Complexity of shift bribery in committee elections. In *Proc. AAAI 2016*, pages 2452–2458, 2016.
- 11 Robert Brederbeck, Piotr Faliszewski, Rolf Niedermeier, and Nimrod Talmon. Large-scale election campaigns: Combinatorial shift bribery. *J. Artificial Intelligence Res.*, 55:603–652, 2016.
- 12 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inform. and Comput.*, 85(1):12–75, 1990.
- 13 Britta Dorn and Ildikó Schlotter. Multivariate complexity analysis of swap bribery. *Algorithmica*, 64(1):126–151, 2012.

- 14 Edith Elkind, Piotr Faliszewski, and Arkadii Slinko. Swap bribery. In *Proc. SAGT 2009*, volume 5814 of *Lecture Notes Comput. Sci.*, pages 299–310, 2009.
- 15 Edith Elkind, Piotr Faliszewski, and Arkadii Slinko. Swap bribery, 2009. URL: <https://arxiv.org/abs/0905.3885>.
- 16 Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 1972.
- 17 Piotr Faliszewski. Nonuniform bribery. In *Proc. AAMAS 2008*, pages 1569–1572, 2008.
- 18 Piotr Faliszewski, Edith Hemaspaandra, and Lane A. Hemaspaandra. How hard is bribery in elections? *J. Artificial Intelligence Res.*, 40:485–532, 2009.
- 19 Piotr Faliszewski, Edith Hemaspaandra, and Lane A. Hemaspaandra. Multimode control attacks on elections. *J. Artificial Intelligence Res.*, 40:305–351, 2011.
- 20 Piotr Faliszewski, Edith Hemaspaandra, Lane A. Hemaspaandra, and Jörg Rothe. Llull and copeland voting computationally resist bribery and constructive control. *J. Artificial Intelligence Res.*, 35:275–341, 2009.
- 21 Robert Ganian and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP. In *Proc. AAAI 2016*, pages 710–716, 2016.
- 22 Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk.  $n$ -fold integer programming in cubic time. *Math. Program.*, 137(1-2, Ser. A):325–341, 2013.
- 23 Hendrik W. Lenstra, Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983.
- 24 Jesus 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.
- 25 Shmuel Onn. Nonlinear discrete optimization. *Zurich Lectures in Advanced Mathematics, European Mathematical Society*, 2010.
- 26 Ildikó Schlotter, Piotr Faliszewski, and Edith Elkind. Campaign management under approval-driven voting rules. In *Proc. AAAI 2011*, pages 726–731, 2011.
- 27 Hobart P. Young. Extending Condorcet’s rule. *J. Econ. Theory*, 16:335–353, 1977.

# A Complexity Dichotomy for Poset Constraint Satisfaction

Michael Kompatscher<sup>\*1</sup> and Trung Van Pham<sup>†2</sup>

1 Theory and Logic Group, Technische Universität Wien, Vienna, Austria  
michael@logic.at

2 Theory and Logic Group, Technische Universität Wien, Vienna, Austria  
pvtrung@logic.at

---

## Abstract

We determine the complexity of all constraint satisfaction problems over partial orders, in particular we show that every such problem is NP-complete or can be solved in polynomial time. This result generalises the complexity dichotomy for temporal constraint satisfaction problems by Bodirsky and Kára. We apply the so called universal-algebraic approach together with tools from model theory and Ramsey theory to prove our result. In the course of this analysis we also establish a structural dichotomy regarding the model theoretic properties of the reducts of the random partial order.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems, F.4.1 Mathematical Logic

**Keywords and phrases** Constraint Satisfaction, Random Partial Order, Computational Complexity, Universal Algebra, Ramsey Theory

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.47

## 1 Motivation and the result

Reasoning about temporal knowledge is a common task in various areas of computer science, including artificial intelligence, scheduling, computational linguistics and operations research. Usually temporal constraints are expressed as collections of relations between time points or time intervals. A typical computational problem is then to determine whether such a collection is satisfiable or not.

A lot of research in this area concerns only linear models of time. In particular there exists a complete complexity classification of satisfiability problems for linear temporal constraints in [8]. However, many times more complex time models are helpful, for instance in the description of distributed and concurrent systems. In his influential paper [21] Lamport suggested to model time or, to be more precise, the precedence relation in such systems by a partial order. Since then a lot research on distributed computing is based on his approach (e.g. [22], [1], [17], [28]). Thus studying the satisfiability of constraints over partial orders is not only of theoretical, but also of practical interest.

The complexity of a small subclass of these computational problems has already been studied in [14]. We will provide a complete classification, showing that every constraint

---

\* The first author as been funded through projects P27600 and I836-N26 of the Austrian Science Fund (FWF).

† The second author has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 257039) and project P27600 of the Austrian Science Fund (FWF).



satisfaction problems over partial orders is either solvable in polynomial time or is NP-complete. In the following we give a more formal definition:

Let  $\Phi$  be a finite set of quantifier-free formulas in the language consisting of a binary relation symbol  $\leq$ . We define Poset-SAT( $\Phi$ ) as the following computational problem:

**Poset-SAT( $\Phi$ ):**

INSTANCE: Variables  $\{x_1, \dots, x_n\}$  and a conjunction  $\psi(x_1, \dots, x_n)$  of formulas that are obtained from formulas  $\phi \in \Phi$ , by substituting the free variables of  $\phi$  by variables from  $\{x_1, \dots, x_n\}$ ;

QUESTION: Is there a partial order  $(A; \leq)$  such that  $\psi(x_1, \dots, x_n)$  is satisfied in  $A$ , i.e.,  $\psi(a_1, \dots, a_n)$  holds for some elements  $a_1, \dots, a_n \in A$ ?

Our main result then states as the following complexity dichotomy:

► **Theorem 1.** *Let  $\Phi$  be a finite set of quantifier-free  $\leq$ -formulas. Then the computational problem Poset-SAT( $\Phi$ ) is in P or NP-complete.*

For the sake of illustration, we give two examples of Poset-SAT problems:

► **Example 2.** Let  $x < y := x \leq y \wedge \neg(y \leq x)$ . An instance of Poset-SAT( $\{<\}$ ) consists of a set of variables  $\{x_1, \dots, x_n\}$  and a conjunction  $\psi$  of formulas  $x_i < x_j$ . An instance is a yes-instance if and only if  $\psi$  does not contain formulas  $x_{i_1} < x_{i_2}, \dots, x_{i_{n-1}} < x_{i_n}, x_{i_n} < x_{i_1}$ . The existence of such cycles in  $\psi$  can be verified in polynomial time. So Poset-SAT( $\{<\}$ ) is in P.

It is not hard to see that an instance of Poset-SAT( $\{<\}$ ) is satisfied in a partial order if and only if it is satisfied in any extension of it to a linear order. However, this is not true for general Poset-SAT problems; let us in the following denote by Temp-SAT( $\Phi$ ) the problem that asks, if there is a *linear* order satisfying a given instance of Poset-SAT( $\Phi$ ). The Temp-SAT problems were completely classified in [8] under the name of *temporal CSPs*. The complexities of Temp-SAT( $\Phi$ ) and Poset-SAT( $\Phi$ ) can be different even for very simple  $\Phi$  as the following example shows:

► **Example 3.** For short let  $x \perp y$  denote the formula  $\neg(x \leq y \vee y \leq x)$  that defines the incomparability relation on partial orders, and let  $x \not\perp y$  denote its negation. By our results, in particular Theorem 17, Poset-SAT( $\{\perp, \not\perp\}$ ) is an NP-complete problem. On the other hand it is easy to see that Temp-SAT( $\{\perp, \not\perp\}$ ) is solvable in polynomial time.

However, every Temp-SAT problem can be stated as a Poset-SAT problem: Let  $\Phi$  be a finite set of quantifier-free order formulas that are, without loss of generality, in conjunctive normal form. Then, for every  $\phi \in \Phi$ , let  $\phi'$  be the formula obtained by exchanging every negative literal  $\neg(x \leq y)$  by  $y < x$  and let  $\Phi'$  be the collection of all such formulas  $\phi'$ . Thus all formulas in  $\Phi'$  are positive in the language consisting of  $<$  and  $\leq$ . It is not hard to prove that Temp-SAT( $\Phi$ ) is the same problem as Poset-SAT( $\Phi'$ ). Hence our complexity classification for Poset-SAT problems can be regarded as a strengthening of the complexity classification of temporal CSPs in [8].

Our complexity classification is based on the classification of reducts of the random partial order in [23] and the techniques from universal algebra and Ramsey theory that have been developed in [12].

An essential part of the proof is to determine all the model-complete cores of reducts of the random partial. This allows us to sift out those problems that are covered by already known complexity classifications. In other similar classification projects, such as the classification of

phylogeny constraint satisfaction problems [6] or CSPs over Henson graphs [9], the authors only had to deal with one model-complete core after this step. In the case of the Poset-SAT problems, however, we saw us confronted with four such model-complete cores. We therefore expected many tedious case distinctions in our proof. But, quite surprisingly, we were able to handle them all in the same way. It turned out that the relation

$$\text{Low}(x, y, z) := (x < y \wedge x \perp z \wedge y \perp z) \vee (x < z \wedge x \perp y \wedge y \perp z)$$

is in a certain sense the only source of NP-completeness in all these cases. The same strategy could be helpful to simplify the proof of future classification results. We will explain the details in Section 5.

## 2 Strategy and structural insight

### 2.1 Poset-SAT( $\Phi$ ) as a constraint satisfaction problem

In this section we translate the Poset-SAT( $\Phi$ ) problems to constraint satisfaction problems (CSPs) of certain relational structures. Let  $\Gamma$  be a structure in a finite relational language  $\tau$ . A first order  $\tau$ -formula is called *primitive positive*, if it is of the form  $\exists x_1, \dots, x_n(\phi_1 \wedge \dots \wedge \phi_k)$ , where all  $\phi_i$  are atomic  $\tau$ -formulas or equations. The *constraint satisfaction problem* of  $\Gamma$ , or short CSP( $\Gamma$ ), is the problem of deciding whether a given primitive positive sentence is true in  $\Gamma$ . CSPs of finite structures are a well-studied topic in complexity theory. Feder and Vardi famously conjectured in [16] that every constraint satisfaction problem of a *finite* structure is either in P or NP-complete, which, under the assumption of  $P \neq NP$  would make finite CSPs to the biggest known class of problems in NP that avoid NP-intermediate problems. However we are going to consider CSPs of structures with *infinite* domain. For infinite CSPs the dichotomy does not hold; all possible complexities can appear, up to polynomial time [4]. However for Poset-SAT, the machinery developed by Bodirsky and Pinsker in [12] will allow us to prove our result.

Every Poset-SAT problem can be restated as a constraint satisfaction problem of a structure that is first-order definable in the *random partial order*  $\mathbb{P} = (P; \leq)$ , a well-known structure in model theory. The random partial order is the unique countable partial order that is both

- *homogeneous*, i.e., every isomorphism between finite substructures of  $\mathbb{P}$  extends to an automorphism of  $\mathbb{P}$ , and
- *universal*, i.e., it contains an isomorphic copy of every finite partial order.

As a countable homogeneous structure in a finite relational language,  $\mathbb{P}$  has several nice properties; in particular the theory of  $\mathbb{P}$  has quantifier elimination and is  $\omega$ -categorical, i.e. it has a unique countable model up to isomorphism. For more model-theoretical background on homogeneous structures we refer to [18].

Let  $\Phi$  now be a given set of quantifier-free  $\leq$ -formulas. We define  $\Gamma_\Phi$  as the relational structure that has  $P$  as a domain and contains for every  $\phi_i \in \Phi$  a relation  $R_i$ , consisting of all the tuples in  $\mathbb{P}$  satisfying  $\phi_i$ . So all the relations of  $\Gamma_\Phi$  are first order definable in  $\mathbb{P}$ . Following an established convention [29, 11] we call  $\Gamma_\Phi$  a *reduct* of  $\mathbb{P}$ . By the universality of  $\mathbb{P}$  it is straightforward to see that Poset-SAT( $\Phi$ ) is essentially the same problem as CSP( $\Gamma_\Phi$ ).

Also the opposite direction holds: By the quantifier elimination of  $\mathbb{P}$ , every reduct  $\Gamma$  of  $\mathbb{P}$  can be defined by a set of quantifier-free formulas  $\Phi_\Gamma$ . It is not hard to see that CSP( $\Gamma$ ) is the same problem as Poset-SAT( $\Phi_\Gamma$ ). So the class of Poset-SAT problems corresponds exactly to the CSPs of reduct of  $\mathbb{P}$ .

Note that the reformulation of Poset-SAT as constraint satisfaction problem would also work with any other universal partial order. However, the choice of the random partial order is not arbitrary, since the concept of homogeneity plays a central role in infinite valued CSPs: On one hand the class of reducts of homogeneous structures is a wide generalisation of the class of finite structures and CSPs of such structures appear often as natural problems in many areas of computer science, e.g. in phylogenetic analysis, computational linguistics, temporal and spatial reasoning and many others (see [4] for reference).

On the other hand homogeneity helps to transfer the algebraic techniques from finite-domain constraint satisfaction problems to infinite structures.

## 2.2 The universal algebraic approach

Our new aim thus is to classify all CSPs of reducts  $\Gamma$  of the random partial order  $\mathbb{P}$ . We use the *universal-algebraic approach* to constraint satisfaction; in this section we explain its basic principles and give an outline of the proof.

A relation  $R$  is *primitive positive definable* or *pp-definable* in  $\Gamma$  if there is a primitive positive formula  $\phi(x_1, \dots, x_n)$  in the language of  $\Gamma$  such that  $(a_1, \dots, a_n) \in R$  if and only if  $\phi(a_1, \dots, a_n)$  holds in  $\Gamma$ . If  $R$  is pp-definable in  $\Gamma$ , the CSP induced by the extended structure  $(\Gamma, R)$  reduces to CSP( $\Gamma$ ) in polynomial time, as Jeavons observed in [19]. So we have to study the reducts of  $\mathbb{P}$  only up to pp-interdefinability.

We say a function  $f : P^n \rightarrow P$  *preserves* a relation  $R \subseteq P^k$  if for all  $\bar{t}_1, \dots, \bar{t}_n \in R$  also the component-wise computed tuple  $f(\bar{t}_1, \dots, \bar{t}_n)$  lies in  $R$ . Otherwise we say  $f$  *violates*  $R$ . By the *polymorphism clone*  $\text{Pol}(\Gamma)$  we denote the set of all finitary functions that preserve all relations of  $\Gamma$ . This set is closed under composition and contains all projections. Furthermore it is closed with respect to the *topology of pointwise convergence*, i.e. the topology on  $\text{Pol}(\Gamma)$  given by basis open neighbourhoods of the form  $\{g : P^n \rightarrow P : g|_A = f|_A\}$  of  $f : P^n \rightarrow P$ , where  $A$  is a finite set. It is well-known that for  $\omega$ -categorical  $\Gamma$  the relations preserved by all elements of  $\text{Pol}(\Gamma)$  are exactly those that are primitive positive definable in  $\Gamma$  (see e.g. [10]). So reducts with the same polymorphism clone induce CSPs of the same complexity. Hence our aim is to understand the polymorphism clones of reducts of  $\mathbb{P}$ .

The tools that we use to study those clones were invented by Bodirsky and Pinsker and used to classify the phylogeny CSPs [6], the CSPs of reducts of the random graph [12], and of all other homogeneous graphs [9]. A key component is the use of Ramsey theory and the concept of *canonical functions*. A short introduction can be found in [26], an extended survey in [4]; we will discuss this usage of Ramsey theory only very briefly in our paper.

We start in Section 3, by classifying the endomorphism monoids of reducts  $\Gamma$  of  $\mathbb{P}$  (in other words we study the unary part of  $\text{Pol}(\Gamma)$ ). This step generalises the main result of [23], which says that there are exactly five automorphism groups of reducts of  $\mathbb{P}$ . By our proof we also identify all the cases in which the complexity is already known from the classification of temporal CSPs in [8]. After that, only CSPs of reducts whose automorphism group lies dense in the endomorphism monoid are left unclassified.

The next proof step is to study in every such case which NP-hard relations can appear, and to prove, with the help of the universal-algebraic approach, that these are the only ones. Since already the proof of [23] was quite involved, it would be reasonable to expect that also our proof splits up in many case distinctions. But quite surprisingly, in Section 5 we are able to reduce several cases to the one where the relations  $<$  and  $\perp$  are primitively positive definable. We can do so by fixing finitely many constants in the reduct and constructing a pp-interpretation of  $(P; <, \perp)$  in this extended structure. This strategy could also be helpful in other CSP classifications.



As in previous results in [8], [12], [6] and [9], our complexity dichotomy corresponds to a structural dichotomy expressible in the language of model theory and universal algebra:

► **Theorem 4.** *Let  $\Gamma$  be a reduct of the random partial order and let  $\Delta$  be its model-complete core. Then exactly one of the following cases applies:*

- *There are polymorphisms  $f$  and endomorphisms  $e_1, e_2$  of  $\Delta$  such that*

$$e_1(f(x, y)) = e_2(f(y, x))$$

*or there are polymorphisms  $f$  and endomorphisms  $e_1, e_2, e_3$  such that*

$$e_1(f(x, x, y)) = e_2(f(x, y, x)) = e_3(f(y, x, x))$$

*holds for all  $x, y \in \Delta$ . In this case  $\text{CSP}(\Gamma)$  is in  $P$ .*

- *An extension of  $\Delta$  by finitely many constants pp-interprets all finite structures. In this case  $\text{CSP}(\Gamma)$  is NP-complete.*

The concept of *pp-interpretability* mentioned in Theorem 4 is a natural generalisation of pp-definability: A *pp-interpretation* of a structure  $\Delta$  in  $\Gamma$  is a surjective partial map  $I : \Gamma^n \rightarrow \Delta$  such that the domain of  $I$  and the preimage of every relation of  $\Delta$  (including equality) are pp-definable in  $\Gamma$ . If there is a pp-interpretation of  $\Delta$  in  $\Gamma$ , then  $\text{CSP}(\Delta)$  reduces to  $\text{CSP}(\Gamma)$  in polynomial time.

By Theorem 4 our result is also in accordance with the algebraic dichotomy conjecture for CSPs over finitely bounded homogeneous structure (see [2]). We will give a proof sketch of Theorem 4 in the following sections; the complete proof can be found in the extended version of this paper, which is available as arXiv preprint [20]. The sections in the extended version correspond to the sections of the same name in the extended paper.

### 3 A preclassification by model-complete cores

By  $\text{Aut}(\Gamma)$  we denote the automorphism group and by  $\text{End}(\Gamma)$  the endomorphism monoid of a structure  $\Gamma$ . Both are also topological objects with respect to the topology of pointwise convergence. A countable structure  $\Gamma$  is called a *model-complete core* if  $\text{Aut}(\Gamma)$  is dense in  $\text{End}(\Gamma)$ . Every countable  $\omega$ -categorical structure  $\Gamma$  is *homomorphically equivalent* to a unique model-complete core  $\Delta$ , meaning that there exists a homomorphism from  $\Gamma$  to  $\Delta$  and vice-versa [3]. Since the CSPs of homomorphically equivalent structures are equal, one can work with model-complete cores instead. In many complexity classification projects it has been proven that working with model-complete cores is more manageable than non-model-complete core structures. Moreover recently a dichotomy complexity conjecture for infinite-domain CSPs has been stated in [2] that, as for finite CSPs, talks about the model-complete cores of a structure.

In this section we sketch the proof of the following proposition, which essentially calculates the model-complete cores of the reducts of the random partial order.

► **Proposition 5.** *Let  $\Gamma$  be a reduct of  $\mathbb{P}$ . Then at least one of the following cases applies:*

1.  *$\text{End}(\Gamma)$  contains a constant function,*
2.  *$\text{End}(\Gamma)$  contains an injection  $g_{<}$  that preserves  $<$  and maps  $P$  onto a chain,*
3.  *$\text{End}(\Gamma)$  contains an injection  $g_{\perp}$  that preserves  $\perp$  and maps  $P$  onto an antichain,*
4. *The automorphism group  $\text{Aut}(\Gamma)$  is dense in  $\text{End}(\Gamma)$ .*

In the first case the model-complete core of  $\Gamma$  contains only one element and  $\text{CSP}(\Gamma)$  is trivial. In the second case, by taking the image of  $\Gamma$  under  $g_{<}$  we obtain a structure that is isomorphic to a reduct of the rational order  $(\mathbb{Q}; <)$  (and homomorphically equivalent to  $\Gamma$ ). Hence  $\text{CSP}(\Gamma)$  belongs to the temporal CSPs classified in [8]. This applies for instance to Example 2 from the introduction, where  $\text{CSP}(P; <)$  is essentially the same problem as  $\text{CSP}(\mathbb{Q}, <)$ . Case 3 similarly allows a reduction of  $\text{CSP}(\Gamma)$  to the CSP of a reduct of  $(\mathbb{Q}; \neq)$ . These CSPs were already studied in [7] and are also part of the classification [8].

By Proposition 5 only the CSPs of the reducts in case 4 are left unclassified. There are exactly five different possible automorphism groups of reducts of  $\mathbb{P}$ . If we turn the random partial order  $\mathbb{P}$  upside-down, the obtained partial order is again isomorphic to  $\mathbb{P}$ . Hence there exists a bijection  $\uparrow: P \rightarrow P$  such that for all  $x, y \in P$  we have  $x < y$  if and only if  $\uparrow(y) < \uparrow(x)$ . By  $\circlearrowleft: P \rightarrow P$  we denote the *rotation on a random filter*, a bijective function introduced in [23] and further studied in [24] that rotates the partial order, in a certain way, around a generic upwards closed set.

Let  $F$  be a subset of the symmetric group of  $P$ . For short let  $\langle F \rangle$  denote the smallest closed group containing  $\text{Aut}(\mathbb{P})$  and the functions in  $F$ . Let  $\overline{F}$  be the topological closure of  $F$  in the topological space of all functions in  $P^P$ . Then the reducts of  $\mathbb{P}$  are classified up to automorphisms by the following result:

► **Theorem 6** (Theorem 1 in [23]). *Let  $\Gamma$  be a reduct of  $\mathbb{P}$ . Then  $\text{Aut}(\Gamma)$  is equal to  $\text{Aut}(\mathbb{P})$ ,  $\langle \uparrow \rangle$ ,  $\langle \circlearrowleft \rangle$  or  $\langle \uparrow, \circlearrowleft \rangle$  or the full symmetric group of  $P$ .*

We define the following relations on  $P$  that can be seen as generalisation of the betweenness relation, the cyclic ordering and the separation relation on the rational order (cf. [15]):

$$\begin{aligned} \text{Betw}(x, y, z) &:= (x < y \wedge y < z) \vee (z < y \wedge y < x), \\ \text{Cycl}(x, y, z) &:= (x < y \wedge y < z) \vee (z < x \wedge x < y) \vee (y < z \wedge z < x) \vee \\ &\quad (x < y \wedge x \perp z \wedge y \perp z) \vee (y < z \wedge x \perp y \wedge x \perp z) \vee (z < x \wedge y \perp z \wedge y \perp x), \\ \text{Sep}(x, y, z, t) &:= (\text{Cycl}(x, y, z) \wedge \text{Cycl}(y, z, t) \wedge \text{Cycl}(x, y, t) \wedge \text{Cycl}(x, z, t)) \vee \\ &\quad (\text{Cycl}(z, y, x) \wedge \text{Cycl}(t, z, y) \wedge \text{Cycl}(t, y, x) \wedge \text{Cycl}(t, z, x)). \end{aligned}$$

These relations allow us to describe the topological closures of the groups in Theorem 6 as endomorphism monoids:

► **Lemma 7.**

1.  $\text{End}(P; <, \perp) = \overline{\text{Aut}(\mathbb{P})}$
2.  $\text{End}(P; \text{Betw}, \perp) = \langle \uparrow \rangle$
3.  $\text{End}(P; \text{Cycl}) = \langle \circlearrowleft \rangle$
4.  $\text{End}(P; \text{Sep}) = \langle \uparrow, \circlearrowleft \rangle$

Note that the topological closure of the symmetric group is the set of injective functions on  $P$ , therefore it contains the function  $g_{\perp}$ . So in order to prove Proposition 5 we need to show that whenever  $\text{End}(\Gamma)$  is not equal to one of the monoids in Lemma 7, it contains a constant function,  $g_{<}$  or  $g_{\perp}$ . This can be proven, similarly to Theorem 6, with the method of canonical functions.

The *type* of a tuple  $\bar{a}$  of elements of  $\Gamma$  is the set of all the first-order formulas  $\phi(\bar{x})$  such that  $\phi(\bar{a})$  holds in  $\Gamma$ . A function from a structure  $\Delta$  to a structure  $\Gamma$  is called *canonical* if it maps tuples of the same type in  $\Delta$  to tuples of the same type in  $\Gamma$ .

There is an extension of  $\mathbb{P}$  by a linear order relation  $\prec$  such that  $x < y$  implies  $x \prec y$  and  $(P; \leq, \prec)$  is a homogeneous structure. We know from [25] that  $(P; \leq, \prec)$  is a *Ramsey*

structure. Without specifying the details, this fact together with results from [13], guarantees the existence of canonical functions with helpful properties for our analysis of endomorphism monoids:

► **Lemma 8** (Lemma 14 in [13]). *Let  $f : P \rightarrow P$  and  $c_1, \dots, c_n \in P$ . Then there exists a function  $g : P \rightarrow P$  such that*

1.  $g$  is generated by  $f$  and  $\text{Aut}(\mathbb{P})$ , i.e.,  $g$  lies in the smallest closed monoid containing  $f$  and  $\text{Aut}(\mathbb{P})$
2.  $g(c_i) = f(c_i)$  for  $i = 1, \dots, n$ .
3. Regarded as a function from  $(\mathbb{P}, \prec, c_1, \dots, c_n)$  to  $\mathbb{P}$ ,  $g$  is canonical.

**Proof sketch of Proposition 5.** Let  $\Gamma$  be a reduct of  $\mathbb{P}$ . Assume for example that  $\text{End}(\Gamma)$  is not contained in  $\overline{\langle \uparrow, \cup \rangle}$ . By Lemma 7 there is a tuple  $\bar{c} \in P^4$  and a function  $f \in \text{End}(\Gamma)$  such that  $\bar{c} \in \text{Sep}$  but  $f(\bar{c}) \notin \text{Sep}$ . By Lemma 8 we can assume that  $f$  is canonical, when seen as a function from  $(P; \leq, \prec, \bar{c})$  to  $(P; \leq)$ . A combinatorial analysis of possible canonical functions shows that then  $f$  generates  $g_{<}$ ,  $g_{\perp}$  or a constant. The details of this analysis of canonical functions are quite technical and left out in this short version of the paper. Following this strategy one can show that, whenever  $\text{End}(\Gamma)$  is not equal to one of the monoids in Lemma 7, it has to contain  $g_{<}$ ,  $g_{\perp}$  or a constant. ◀

#### 4 The case where $<$ and $\perp$ are pp-definable

In this section we consider all reducts of  $\mathbb{P}$  whose endomorphism monoids are equal to  $\overline{\text{Aut}(\mathbb{P})}$ , trying to identify NP-hard relations that can appear. The following lemma will be of help in this analysis.

► **Lemma 9** (Bodirsky, Kára [8]). *Let  $\Gamma$  be a relational structure and let  $R \subseteq D^k$  be a union of at most  $m$  orbits (i.e. minimal invariant sets) of the component-wise action of  $\text{Aut}(\Gamma)$  on  $D^k$ . If  $\Gamma$  has a polymorphism  $f$  that violates  $R$ , then  $\Gamma$  also has an at most  $m$ -ary polymorphism that violates  $R$ .*

By Lemma 9 the relations  $<$  and  $\perp$  are primitive positive definable in a reduct  $\Gamma$  of  $\mathbb{P}$  if and only if  $\text{End}(\Gamma) = \overline{\text{Aut}(\mathbb{P})}$ .

It is straightforward to see that  $\text{CSP}(P; <, \perp)$  and  $\text{CSP}(P; \leq, \perp)$  both are in P. They are part of a bigger class of CSPs that can be computed in polynomial time. Let  $e_{<} : P^2 \rightarrow P$  be an embedding of the structure  $(P; <)^2$  into  $(P; <)$ ; in other words  $e_{<}$  is an injective function such that  $e_{<}(x, y) < e_{<}(x', y')$  if and only if  $x < y$  and  $x' < y'$ . Similarly let  $e_{\leq}$  be an embedding of  $(P; \leq)^2$  into  $(P; \leq)$ . A more general result in [5] implies that these two operations can be used to characterised so-called *Horn-tractable* structures:

► **Lemma 10.** *Let  $\Gamma$  be a reduct of  $\mathbb{P}$ . Suppose that  $e_{\leq} \in \text{Pol}(\Gamma)$ . Then  $\text{CSP}(\Gamma)$  is in P and every relation in  $\Gamma$  is equivalent to a conjunction of Horn formulas in  $(P; \leq)$ , i.e. formulas of the form:*

$$\begin{aligned} x_{i_1} \leq x_{j_1} \wedge x_{i_2} \leq x_{j_2} \wedge \dots \wedge x_{i_k} \leq x_{j_k} &\rightarrow x_{i_{k+1}} \leq x_{j_{k+1}} \text{ or} \\ x_{i_1} \leq x_{j_1} \wedge x_{i_2} \leq x_{j_2} \wedge \dots \wedge x_{i_k} \leq x_{j_k} &\rightarrow \perp. \end{aligned}$$

*Suppose that  $e_{<} \in \text{Pol}(\Gamma)$ . Then  $\text{CSP}(\Gamma)$  is in P and every relation in  $\Gamma$  is equivalent to a conjunction of Horn formulas in  $(P; <)$ , i.e. formulas of the form:*

$$\begin{aligned} x_{i_1} \triangleleft_1 x_{j_1} \wedge x_{i_2} \triangleleft_2 x_{j_2} \wedge \dots \wedge x_{i_k} \triangleleft_k x_{j_k} &\rightarrow x_{i_{k+1}} \triangleleft_{k+1} x_{j_{k+1}} \text{ or} \\ x_{i_1} \triangleleft_1 x_{j_1} \wedge x_{i_2} \triangleleft_2 x_{j_2} \wedge \dots \wedge x_{i_k} \triangleleft_k x_{j_k} &\rightarrow \perp, \end{aligned}$$

where  $\triangleleft_i \in \{<, =\}$  for all  $i = 1, \dots, k + 1$ .

A polynomial time algorithm for these Horn-tractable structures can be constructed similarly to the algorithm for Horn clauses in Boolean SAT: By resolution, one can substitute the input in polynomial time by a set of positive literals. The satisfiability of these literals can then also be checked in polynomial time, as in Example 2.

We are now going to show that every reduct  $\Gamma$  of  $\mathbb{P}$  with  $\text{End}(\Gamma) = \overline{\text{Aut}(\mathbb{P})}$  that does not have  $e_<$  or  $e_\leq$  as polymorphism induces an NP-complete CSP. Let us define the relation

$$\text{Low}(x, y, z) := (x < y \wedge x \perp z \wedge y \perp z) \vee (x < z \wedge x \perp y \wedge y \perp z).$$

It is easy to see that every endomorphism of  $\text{Low}$  is injective and that  $\text{Low}$  is not preserved by  $e_<$  or  $e_\leq$ . In fact  $\text{Low}$  can be used to characterise all reducts that do not have  $e_<$  or  $e_\leq$  as polymorphisms:

- **Proposition 11.** *Let  $\Gamma$  be a reduct of  $\mathbb{P}$  such that  $\text{End}(\Gamma) = \overline{\text{Aut}(\mathbb{P})}$ . Then either*
- *the relation  $\text{Low}$  is pp-definable in  $\Gamma$  or,*
  - *one of the functions  $e_<$ ,  $e_\leq$  is a polymorphism of  $\Gamma$ .*

The proof of this result is quite technical and makes up a large part of the long version of this paper. Here we only give a short proof sketch.

**Proof sketch of Proposition 11.** Assume that  $\text{Low}$  is not pp-definable in  $\Gamma$ . By Lemma 9 there has to be a binary  $f \in \text{Pol}(\Gamma)$  that violates  $\text{Low}$ . Using the homogeneity of  $\mathbb{P}$  we can even further assume that there are elements  $a, b, c \in P$  such that  $(a, b, c) \in \text{Low}$ , but  $(f(a, a), f(b, c), f(c, b)) \notin \text{Low}$ . We now make again use of canonical functions, however, with the following generalisation of Lemma 8:

► **Lemma 12** (Lemma 21 in [13]). *Let  $f : P^r \rightarrow P$  and  $c_1, \dots, c_n \in P$ . Then there exists a function  $g : P^r \rightarrow P$  such that*

1.  *$g$  is generated by  $f$  and  $\text{Aut}(\mathbb{P})$ , i.e. lies in the smallest closed clone containing  $f$  and  $\text{Aut}(\mathbb{P})$ ;*
2. *the restriction of  $g$  to  $\{c_1, \dots, c_n\}^r$  is equal to  $f$ ;*
3. *regarded as a function from  $(\mathbb{P}, <, c_1, \dots, c_n)^r$  to  $\mathbb{P}$ ,  $g$  is canonical.*

By Lemma 12 we can assume that  $f$  is canonical, when seen as a function from the structure  $(P; \leq, <, a, b, c)^2$  to  $(P; \leq)$ . An analysis of all possible such canonical functions shows that  $f$  together with  $\text{End}(\Gamma)$  has to generate  $e_<$  or  $e_\leq$ . For reasons of space, we omit this combinatorial analysis here; it can be found in Section 7 of the extended paper [20]. ◀

We remark that one can even show that every  $f : P^2 \rightarrow P$  preserving  $\text{Low}$  has to be *dominated*, meaning that  $x < x'$  implies  $f(x, y) < f(x', y')$  and  $x \perp x'$  implies  $f(x, y) \perp f(x', y')$ , or that  $f$  satisfies the symmetric conditions for its second coordinate. This fact is proven in the extended version of the paper. It only remains to show that  $\text{CSP}(P; \text{Low})$  is NP-complete to prove the complexity dichotomy for the case where  $<$  and  $\perp$  are pp-definable.

The Boolean 3-satisfiability problem is a well-studied NP-complete problem [27]. It can be written as  $\text{CSP}(\{0, 1\}; \text{XOR}, \text{3OR})$  with  $\text{XOR} = \{(0, 1), (1, 0)\}$  and  $\text{3OR} = \{0, 1\}^3 \setminus \{(0, 0, 0)\}$ . In practice we often show the NP-hardness of  $\text{CSP}(\Gamma)$  by finding a pp-interpretation of  $(\{0, 1\}; \text{XOR}, \text{3OR})$  or another Boolean NP-complete structure in  $\Gamma$ . We remark that if such a pp-interpretation exists, *every* finite structure has a pp-interpretation in  $\Gamma$ .

Also adding constants does not change the complexity of a CSP, by the following lemma:

- **Lemma 13** (Corollary 3.6.25 in [4]). *Let  $\Gamma$  be an  $\omega$ -categorical model-complete core and  $c_1, c_2, \dots, c_k$  be constants of  $\Gamma$ . Then  $\text{CSP}(\Gamma)$  is polynomially equivalent to  $\text{CSP}(\Gamma, c_1, c_2, \dots, c_k)$ .*

Thus to prove the NP-hardness of a CSP of a  $\omega$ -categorical model-complete core we often add sufficient constants to the structure and then construct a pp-interpretation of an NP-hard structure in the resulting structure.

► **Lemma 14.** *Let  $a, b \in P$  be constants such that  $a \perp b$ . Then there is a primitive positive interpretation of  $(\{0, 1\}; \text{XOR}, \text{3OR})$  in  $(P; \text{Low}, a, b)$ . Hence  $\text{CSP}(P; \text{Low})$  is NP-complete.*

**Proof.** Let  $D := \{x \in P : \text{Low}(x, a, b)\}$ ,  $D_0 := \{x \in D : x < a\}$  and  $D_1 := \{x \in D : x < b\}$ . We define  $I : D \rightarrow \{0, 1\}$  by setting  $I(x) = 0$  if  $x \in D_0$  and  $I(x) = 1$  if  $x \in D_1$ . We claim that  $I$  is a pp-interpretation of  $(\{0, 1\}; \text{XOR}, \text{3OR})$  in  $(P; \text{Low}, a, b)$ .

Clearly the domain  $D$  and also the sets  $D_0$  and  $D_1$  are pp-definable in  $(P; \text{Low}, a, b)$ . Let

$$\text{Abv}(x, y, z) := (y < x \wedge x \perp z \wedge y \perp z) \vee (z < x \wedge x \perp y \wedge z \perp y)$$

be the, in a sense, dual of  $\text{Low}$ . The relation  $\text{Abv}$  is not preserved by  $e_<$  and  $e_\leq$ . By Proposition 11 we know that  $\text{Low}$  is pp-definable in  $\text{Abv}$ . For symmetry reasons also  $\text{Abv}$  is pp-definable in  $\text{Low}$ . The equation  $I(x) = I(y)$  holds if and only if  $\exists z \text{Abv}(b, x, z) \wedge \text{Abv}(b, y, z)$ . Hence the preimage of equality under  $I$  is pp-definable in  $(P; \text{Low})$ . Furthermore

$$(I(x), I(y)) \in \text{XOR} \leftrightarrow \exists u, v \in D (I(x) = I(u) \wedge I(x) = I(v) \wedge \text{Abv}(a, u, v) \wedge \text{Abv}(b, u, v)).$$

Let  $R(x, y, z, t) := \exists u \text{Abv}(u, y, z) \wedge \text{Abv}(x, u, t)$ . Then  $R(x, y, z, t)$  implies that  $x$  is greater than at least one of the elements  $\{y, z, t\}$ . One can show the following equivalence:

$$I(x_1, x_2, x_3) \in \text{3OR} \leftrightarrow \exists y_1, y_2, y_3 \in D (R(a, y_1, y_2, y_3) \wedge \bigwedge_{i=1}^3 I(x_i) = I(y_i)).$$

So  $I$  is a pp-interpretation of  $(\{0, 1\}; \text{XOR}, \text{3OR})$  in  $(P; \text{Low}, a, b)$  and  $\text{CSP}(P; \text{Low}, a, b)$  is NP-complete. Adding finitely many constants to a model-complete core does not increase the complexity of the induced constraint satisfaction problem by a result in [3]. So  $\text{CSP}(P; \text{Low})$  is NP-complete. ◀

As an immediate consequence we get the following corollary of Proposition 11:

- **Corollary 15.** *Let  $\Gamma$  be a reduct of  $\mathbb{P}$  such that  $\text{End}(\Gamma) = \overline{\text{Aut}(\mathbb{P})}$ . Then*
- *either the relation  $\text{Low}$  is pp-definable in  $\Gamma$  and  $\text{CSP}(\Gamma)$  is NP-complete,*
  - *or one of the functions  $e_<$ ,  $e_\leq$  is a polymorphism of  $\Gamma$  and  $\text{CSP}(\Gamma)$  is in  $P$ .*

## 5 Hardness of Betw, Cycl and Sep

By Proposition 5 we are left with the reducts  $\Gamma$  of  $\mathbb{P}$  whose endomorphism monoids are equal to  $\overline{\langle \updownarrow \rangle}$ ,  $\overline{\langle \circ \rangle}$  or  $\overline{\langle \updownarrow, \circ \rangle}$ . As it turns out, in all these cases the induced CSPs are NP-hard. Interestingly, we can prove this fact for all three cases in the same way: After fixing finitely many constants, which is a feasible reduction by Lemma 13, we can construct a pp-interpretation of  $(P; <, \text{Low})$  in the structure. To be more precise the following holds:

► **Proposition 16.** *Let  $\Gamma$  be a reduct of  $\mathbb{P}$  such that  $\text{End}(\Gamma)$  is equal to  $\overline{\langle \updownarrow \rangle}$ ,  $\overline{\langle \circ \rangle}$  or  $\overline{\langle \updownarrow, \circ \rangle}$ . Then there are elements  $c_1, \dots, c_n \in P$  and a subset  $X \subseteq P$  such that*

- *$(X; \leq)$  is isomorphic to  $\mathbb{P}$ ;*
- *$X$  is pp-definable in  $(\Gamma, c_1, \dots, c_n)$ ;*
- *the restrictions of  $<$  and  $\text{Low}$  to  $X$  are pp-definable in  $(\Gamma, c_1, \dots, c_n)$ .*

*Hence the identity mapping on  $X$  is a pp-interpretation of  $(P; <, \text{Low})$  in  $\Gamma$ , and  $\text{CSP}(\Gamma)$  is NP-complete.*

**Proof.** We demonstrate the proof only for the case, where  $\text{End}(\Gamma) = \overline{\langle \cup \rangle}$ . The other cases are discussed in the extended paper [20]. Note that the relation  $\text{Cycl}$  is an orbit of  $\langle \cup \rangle$  on  $P^3$ . So Lemma 9 implies that  $\text{Cycl}$  is primitive positive definable in  $\Gamma$ . Without loss of generality let  $\Gamma = (P; \text{Cycl})$ .

Let  $c, d$  be two constants in  $P$  such that  $c < d$  and let  $X := \{x \in P : c < x < d\} = \{x \in P : \text{Cycl}(c, x, d)\}$ . By a back-and-forth argument it is easy to show that  $\mathbb{P}$  and  $(X; \leq)$  are isomorphic. For  $x, y \in X$  we have that  $x < y$  is equivalent to  $\text{Cycl}(c, x, y)$ . The incomparability relation  $x \perp y$  is also pp-definable in  $X$  as follows:

$$\begin{aligned} \exists a, b, c, d (x < a < c \wedge x < b < d \wedge y < c \wedge y < d) \wedge \\ (\text{Cycl}(x, a, y) \wedge \text{Cycl}(x, b, y) \wedge \text{Cycl}(y, c, b) \wedge \text{Cycl}(y, d, a) \wedge \text{Cycl}(b, d, c) \wedge \text{Cycl}(a, c, d)). \end{aligned}$$

The two maps  $e_{<} : P^2 \rightarrow P$  and  $e_{\leq} : P^2 \rightarrow P$  do not preserve  $\text{Cycl}$ . By Proposition 11 we have that  $\text{Low}$  is pp-definable in  $(P; <, \perp, \text{Cycl})$ . So the identity on  $X$  gives us a pp-interpretation of  $(P; \text{Low})$  in  $(P; \text{Cycl}, c, d)$ , which concludes the proof.  $\blacktriangleleft$

We observed that in the already existing complexity classifications, certain proof parts could have been simplified using the same principle, i.e. fixing finitely many constants in a model-complete core reduct, to obtain the relations of the underlying homogeneous structure.

So it is natural to ask, if Proposition 16 works in general: Let  $\Delta$  be a homogeneous structure and  $\Gamma$  be a reduct of  $\Delta$  that is a model-complete core. Can we then extend  $\Delta$  by finitely many constants  $c_1, \dots, c_n$  such that the identity mapping on a definable subset of  $(\Gamma, c_1, \dots, c_n)$  pp-interprets  $\Delta$ ? The answer to this question is negative: A non-trivial counterexample is given by the random ordered graph  $\Delta = (G; E, <)$  and the random tournament  $\Gamma = (G; T)$  defined by  $T(x, y) := x \neq y \wedge (x < y \leftrightarrow E(x, y))$ .

## 6 Classification

We sum up the results of the last two sections:

► **Theorem 17.** *Let  $\Gamma$  be a reduct of  $\mathbb{P}$  in a finite relational language and a model-complete core. Then exactly one of the following two cases holds:*

- *One of the relations  $\text{Low}$ ,  $\text{Betw}$ ,  $\text{Cycl}$ ,  $\text{Sep}$  is pp-definable in  $\Gamma$ . An extension of  $\Gamma$  by finitely many constants pp-interprets all finite structures and  $\text{CSP}(\Gamma)$  is NP-complete.*
- *$\text{Pol}(\Gamma)$  contains  $e_{<}$  or  $e_{\leq}$  and  $\text{CSP}(\Gamma)$  is in P.*

**Proof.** If  $\text{Low}$  is pp-definable in  $\Gamma$ , the statement holds by Lemma 14. Next, assume that one of the relations  $\text{Betw}$ ,  $\text{Cycl}$  or  $\text{Sep}$  is pp-definable in  $\Gamma$ . By Proposition 16 we have a pp-interpretation of  $(P; \text{Low})$  in  $\Gamma$ . By the transitivity of pp-interpretations and Lemma 14 we can obtain a pp-interpretation of  $(\{0, 1\}; \text{XOR}, 3\text{OR})$  in  $\Gamma$ , extended by finitely many constants. Hence all finite structure are pp-interpretable in an extension of  $\Gamma$  by finitely many constants. At last, assume that  $\text{Low}$ ,  $\text{Betw}$ ,  $\text{Cycl}$ ,  $\text{Sep}$  are not pp-definable in  $\Gamma$ . By Proposition 5 we have  $\text{End}(\Gamma) = \overline{\text{Aut}(\mathbb{P})}$  and therefore, by Corollary 15,  $e_{<}$  or  $e_{\leq}$  is a polymorphism of  $\Gamma$  and  $\text{CSP}(\Gamma)$  is in P.  $\blacktriangleleft$

Theorem 17 allows us to prove our main result.

**Proof of Theorem 4.** By Proposition 5 we know that the model-complete core of  $\Gamma$  is equal to a one-element set, a reduct of  $(\mathbb{Q}, <)$  or to  $\Gamma$  itself. In the first two cases the dichotomy in Theorem 4 holds by the analysis in [8], see also Theorem 10.1.1. in [4]. If  $\Gamma$  itself is a model-complete core, the dichotomy holds by Theorem 17 and the observation that

there are endomorphisms  $\alpha, \alpha' \in \text{End}(\mathbb{P})$  such that  $e_{<}(x, y) = \alpha(e_{<}(y, x))$  respectively  $e_{\leq}(x, y) = \alpha'(e_{\leq}(y, x))$ .  $\blacktriangleleft$

We identified the problems  $\text{Poset-SAT}(\Phi)$  with CSPs of reducts  $\Gamma$  of  $\mathbb{P}$ , so also the complexity dichotomy for  $\text{Poset-SAT}(\Phi)$  in Theorem 1 is true.

We remark that also the “meta-problem” of deciding if  $\text{Poset-SAT}(\Phi)$  is NP-complete for a given finite set of formulas  $\Phi$ , is decidable. The main result in [13] implies that determining whether a quantifier-free formula in  $\mathbb{P}$  is pp-definable in a given reduct  $\Gamma$  of  $\mathbb{P}$ , is a decidable problem. Hence it is also decidable to tell if  $\Gamma$  is a model-complete core. If  $\Gamma$  is a model-complete core, the list of NP-hard relations in Theorem 17 allows us also to decide whether  $\text{CSP}(\Gamma)$  is NP-complete. A similar list exists for the case where the model-complete core of  $\Gamma$  is a reduct of the rational order in [8].

**Acknowledgements.** We would like to thank Michael Pinsker for his many helpful comments on this paper. Also we would like to thank him and Manuel Bodirsky for giving us counterexamples showing that Proposition 16 cannot be generalised to all homogeneous structures.

---

## References

- 1 Frank D. Anger. On Lamport’s interprocessor communication model. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 11(3):404–417, 1989.
- 2 Libor Barto and Michael Pinsker. The algebraic dichotomy conjecture for infinite domain constraint satisfaction problems, 2016. preprint, arXiv:1602.04353, to appear in the proceedings of LICS 2016.
- 3 Manuel Bodirsky. The core of a countably categorical structure. In *STACS 2005*, pages 110–120. Springer, 2005.
- 4 Manuel Bodirsky. Complexity classification in infinite-domain constraint satisfaction, 2012. Mémoire HDR à l’Université Paris 7, arXiv:1201.0856v7.
- 5 Manuel Bodirsky, Hubie Chen, Jan Kára, and Timo von Oertzen. Maximal infinite-valued constraint languages. *Theoret. Comput. Sci.*, 410(18):1684–1693, 2009.
- 6 Manuel Bodirsky, Peter Jonsson, and Trung Van Pham. The complexity of phylogeny constraint satisfaction. In *STACS 16*, 2016.
- 7 Manuel Bodirsky and Jan Kára. The complexity of equality constraint languages. *Theory of Computing Systems*, 43(2):136–158, 2008.
- 8 Manuel Bodirsky and Jan Kára. The complexity of temporal constraint satisfaction problems. *Journal of the ACM*, 57(2):Art. 9, 41, 2010.
- 9 Manuel Bodirsky, Barnaby Martin, Michael Pinsker, and András Pongrácz. Constraint satisfaction problems for reducts of homogeneous graphs, 2016. Preprint, arXiv:1602.05819.
- 10 Manuel Bodirsky and Jaroslav Nešetřil. Constraint satisfaction with countable homogeneous templates. *Journal of Logic and Computation*, 16(3):359–373, 2006.
- 11 Manuel Bodirsky and Michael Pinsker. Reducts of Ramsey structures. *AMS Contemporary Mathematics, vol. 558 (Model Theoretic Methods in Finite Combinatorics)*, pages 489–519, 2011.
- 12 Manuel Bodirsky and Michael Pinsker. Schaefer’s theorem for graphs. *J. ACM*, 62(3):19:1–19:52, 2015.
- 13 Manuel Bodirsky, Michael Pinsker, and Todor Tsankov. Decidability of definability. *Journal of Symbolic Logic*, 78(4):1036–1054, 2013.
- 14 Mathias Broxvall and Peter Jonsson. Point algebras for temporal reasoning: Algorithms and complexity. *Artificial Intelligence*, 149(2):179–220, 2003.

- 15 Peter Cameron. Transitivity of permutation groups on unordered sets. *Mathematische Zeitschrift*, 148:127–139, 1976.
- 16 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. *SIAM Journal on Computing*, 28:57–104, 1999.
- 17 Patrice Godefroid, J. van Leeuwen, J. Hartmanis, G. Goos, and Pierre Wolper. *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*, volume 1032. Springer Heidelberg, 1996.
- 18 Wilfrid Hodges. *A shorter model theory*. Cambridge University Press, Cambridge, 1997.
- 19 Peter Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200(1):185–204, 1998.
- 20 Michael Kompatscher and Trung Van Pham. A complexity dichotomy for poset constraint satisfaction, 2016. Preprint, arXiv:1603.00082.
- 21 Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.
- 22 Leslie Lamport. The mutual exclusion problem: Part I – A theory of interprocess communication. *Journal of the ACM*, 33(2):313–326, 1986.
- 23 Péter Pál Pach, Michael Pinsker, Gabriella Pluhár, András Pongrácz, and Csaba Szabó. Reducts of the random partial order. *Advances in Mathematics*, 267:94–120, 2014.
- 24 Péter Pál Pach, Michael Pinsker, András Pongrácz, and Csaba Szabó. A new operation on partially ordered sets. *Journal of Combinatorial Theory, Series A*, 120:1450–1462, 2013.
- 25 Madelaine Paoli, William T. Trotter, and James W. Walker. Graphs and orders in Ramsey theory and in dimension theory. In *Graphs and Order*, pages 351–394. Springer, 1985.
- 26 Michael Pinsker. Algebraic and model theoretic methods in constraint satisfaction, 2015. preprint, arXiv:1507.00931.
- 27 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226. ACM, 1978.
- 28 Andrew S. Tanenbaum and Maarten Van Steen. *Distributed systems*. Prentice-Hall, 2007.
- 29 Simon Thomas. Reducts of the random graph. *The Journal of symbolic logic*, 56(01):176–181, 1991.



# Structural Properties and Constant Factor-Approximation of Strong Distance- $r$ Dominating Sets in Sparse Directed Graphs\*

Stephan Kreutzer<sup>1</sup>, Roman Rabinovich<sup>2</sup>, Sebastian Siebertz<sup>3</sup>, and  
Grischa Weberstädt<sup>4</sup>

1 Technische Universität Berlin, Berlin, Germany  
stephan.kreutzer@tu-berlin.de

2 Technische Universität Berlin, Berlin, Germany  
roman.rabinovich@tu-berlin.de

3 Technische Universität Berlin, Berlin, Germany  
sebastian.siebertz@tu-berlin.de

4 Technische Universität Berlin, Berlin, Germany  
grischa.weberstaedt@campus.tu-berlin.de

---

## Abstract

Bounded expansion and nowhere dense graph classes, introduced by Nešetřil and Ossona de Mendez [26, 27], form a large variety of classes of uniformly sparse graphs which includes the class of planar graphs, actually all classes with excluded minors, and also bounded degree graphs. Since their initial definition it was shown that these graph classes can be defined in many equivalent ways: by generalised colouring numbers, neighbourhood complexity, sparse neighbourhood covers, a game known as the splitter game, and many more.

We study the corresponding concepts for directed graphs. We show that the densities of bounded depth directed minors and bounded depth topological minors relate in a similar way as in the undirected case. We provide a characterisation of bounded expansion classes by a directed version of the generalised colouring numbers. As an application we show how to construct sparse directed neighbourhood covers and how to approximate directed distance- $r$  dominating sets on classes of bounded expansion. On the other hand, we show that linear neighbourhood complexity does not characterise directed classes of bounded expansion.

**1998 ACM Subject Classification** G.2.1 Combinatorics, G.2.2 Graph Theory

**Keywords and phrases** Directed Graph Structure Theory, Bounded Expansion, Generalised Colouring Numbers, Splitter Game, Approximation Algorithms, Dominating Set

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.48

## 1 Introduction

Structural graph theory provides a wealth of tools and concepts that have proved to be very powerful in the study of approximation, parameterized or classical polynomial time algorithms for common NP-hard graph problems. The algorithmic properties of classes of graphs of *bounded tree width*, of *bounded genus* – especially *planar graphs* – or which exclude a fixed (*topological*) *minor* have been very well studied in the literature and powerful and

---

\* Stephan Kreutzer, Roman Rabinovich and Sebastian Siebertz’s research has been supported by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (ERC Consolidator Grant DISTRUCT, grant agreement No 648527).



very general algorithmic techniques for solving NP-hard problems on any of these classes of graphs have emerged.

Initially, many graph theoretical concepts studied in this area were based on topological aspects of graphs, such as tree width or excluded minors. In [26], Nešetřil and Ossona de Mendez introduced the concept of classes of *bounded expansion* which properly generalise classes of graphs excluding a fixed minor. Bounded expansion classes are defined in terms of edge densities of bounded depth minors. Since their initial definition in [26] it was shown that the concept of graph classes of bounded expansion as well as their generalisation to nowhere dense classes of graphs can equivalently be defined in many other ways: by *generalised colouring numbers* [38], *low tree depth colourings* [26], bounded *neighbourhood complexity* [33], *sparse neighbourhood covers* [15, 16, 28] and a game known as the *splitter game* [16]. This indicates that bounded expansion is a natural concept appearing frequently in different contexts. This intuition is supported for instance by [32] where it was shown that many types of real-world networks indeed are of bounded expansion.

One important consequence of the large number of different characterisations of bounded expansion classes is that every new characterisation developed in the literature also provides a different set of algorithmic techniques that can be used for different types of problems. This has led to a growing number of algorithmic results on bounded expansion classes for parameterized algorithms [8, 12, 16, 29], approximation algorithms [11], kernelization [9, 13] and others.

Starting with Johnson, Robertson, Seymour and Thomas' introduction of *directed tree width* [17], several attempts have been made to generalise the successful theory of algorithmic graph structure theory to the world of directed graphs. Most of these proposals were again based on generalising topological properties, especially bounded tree width, to directed graphs [1, 3, 4, 5, 25, 30, 34]. However, it was subsequently shown that many NP- or W[1]-hard computational problems for directed graphs remain intractable on classes of bounded directed tree width [21, 24]. In fact, it was even claimed in the literature that there cannot be any algorithmically useful digraph width measure [14]. One important reason for these hardness results is that these problems remain intractable even on acyclic or nearly acyclic digraphs and most of proposed width measures have small width on acyclic digraphs.

To overcome these problems, Kreutzer and Tazari [23] initiated the study of generalisations of bounded expansion and nowhere dense classes of graphs to the directed setting. In [23] they studied a concept called *nowhere crownful* classes of digraphs, proved an equivalent characterisation of these classes by *uniformly quasi-wideness* and showed that a variant of the directed dominating set problem becomes fixed-parameter tractable on nowhere crownful classes of digraphs. In the same paper, they defined the concept of *directed bounded expansion*, but did not study it any further. The main and decisive difference between these new approaches and the previous proposals of directed width measures is that nowhere crownful and bounded expansion classes of digraphs do not contain the class of acyclic digraphs as “low width” classes. Quite the contrary, they were specifically designed to distinguish between “easy” instances of acyclic digraphs and computationally hard ones.

**Contributions of this paper.** The focus of this paper is to study in depth classes of digraphs of bounded expansion. Intuitively, a class of digraphs has *bounded expansion* if there is a function  $f: \mathbb{N} \rightarrow \mathbb{N}$  such that for all  $G \in \mathcal{C}$  and all  $r \geq 1$ , every *depth- $r$  minor*  $H$  of  $G$  has edge density  $\frac{|E(H)|}{|V(H)|} \leq f(r)$ . While there is a clear and generally accepted concept of undirected minor, there are various definitions of directed minors that have been studied in the literature, among them *butterfly* minors [17] but also *directed minors* as used in [23]

to define nowhere crownful classes. In Section 3 we show that the concept of bounded expansion of digraphs is irrespective of the notion of directed minor used. In particular, a class has bounded expansion with respect to the directed minors in [23] if, and only if, it has bounded expansion with respect to *topological minors*, i.e. subdivisions. As subdivisions are a canonical notion even in the directed setting, this yields a natural and undisputed definition of bounded expansion.

We show next that classes of bounded expansion can also be characterised by generalised colouring numbers (see Section 4). Generalised colouring numbers have proved to be an extremely fruitful concept on undirected graphs, both structurally and algorithmically. The characterisation of directed bounded expansion by colouring numbers allows us to employ similar algorithmic techniques now also in the directed setting. We show in Section 6 that bounded expansion classes admit constant step winning strategies in a splitter game (see Section 6) and also admit sparse neighbourhood covers (see Section 7).

The dominating set problem is one of the best studied NP-hard graph problems in the literature and has served as benchmark for many different algorithmic techniques. It remains NP-complete on many classes of graphs and is notoriously hard to approximate. In fact, Raz and Safra [31] showed that approximating the domination number within any factor better than  $O(\log n)$  is already NP-hard. Computing dominating sets, or the distance- $r$  variant of it, is a graph theoretical abstraction of many real-life problems such as distributing facilities such as routers or distribution centres to cover a given area or many other similar problems. One of the main applications of dominating sets is to choose a small number of positions in a graph or network so that every vertex in the graph can communicate with a member of the dominating set. The radius  $r$  thereby determines the range that can be covered by a single element of the dominating set.

Motivated by the application above where we want to choose vertices so that every vertex of the network can communicate with an element of the dominating set within distance  $r$ , we study the *strong distance- $r$  dominating set* problem. Here we are asked to find a minimal set  $X$  of vertices in a digraph  $G$  such that every vertex  $v \in V(G)$  is contained together with some element of  $X$  on a closed directed walk of length at most  $2r$ .<sup>1</sup> We show in Section 8 that for every  $r \geq 1$  there is a constant factor approximation algorithm for the strong  $r$  dominating set problem on any class of digraphs of bounded expansion.

## 2 Background from graph theory

In this section we fix our notation. We refer to [2] for background on digraph theory.

**Digraphs, walks and neighbourhoods.** A *digraph*  $G$  consists of a set  $V(G)$  of *vertices* and a set  $E(G) \subseteq V(G) \times V(G)$  of *arcs*. We assume that a digraph  $G$  has no loops, i.e. no edges of the form  $(v, v)$  for  $v \in V(G)$ . A *walk of length  $k$*  in a digraph  $G$  is a sequence  $W = v_0, \dots, v_k$  of vertices of  $G$  such that for each  $0 \leq i < k$  there is an edge  $(v_i, v_{i+1}) \in E(G)$ . A walk is *closed* if  $v_0 = v_k$ , and *open* otherwise. If  $W$  is open, then vertex  $v_0$  is the *initial vertex* of  $W$ ,

<sup>1</sup> A different, perhaps more natural variant would be to require that for every vertex  $v \in V(G)$  there is an  $x \in X$  such that  $G$  contains a directed path of length at most  $r$  from  $x$  to  $v$  and a directed path of length at most  $r$  from  $v$  to  $x$ . We have opted for our notion of strong domination as it fits more nicely with the concept of strong neighbourhoods in Section 2. But the same algorithmic techniques we use in Section 8 could also be used to design a constant factor approximation algorithm for this different definition of strong domination.

vertex  $v_k$  is its *terminal vertex*, and  $v_0$  and  $v_k$  are *end-vertices* of  $W$ . If all vertices of  $W$  are distinct, then  $W$  is a *path* from  $v_0$  to  $v_k$ .

Let  $G$  be a digraph, let  $v \in V(G)$  and let  $r \geq 0$ . The *r-out-neighbourhood* of  $v$ , denoted 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 path of length at most  $r$  from  $v$  to  $u$ . We write  $N^+(v)$  for  $N_1^+(v) \setminus \{v\}$ .

The *r-in-neighbourhood* of  $v$ , denoted 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 path of length at most  $r$  from  $u$  to  $v$ . We write  $N^-(v)$  for  $N_1^-(v) \setminus \{v\}$ .

The *r-strong-neighbourhood* of  $v$ , denoted by  $\tilde{N}_{G,r}(v)$ , or just  $\tilde{N}_r(v)$  if  $G$  is understood, is defined as the set of vertices  $u$  in  $G$  such that  $G$  contains a closed walk of length at most  $2r$  containing  $u$  and  $v$ .

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.

If the edge 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  $(u, v) \in E(G)$  it holds that  $(u, v) \in E(\bar{G})$  and  $(v, u) \in E(\bar{G})$ . Note that  $|E(G)| \leq |E(\bar{G})| \leq 2|E(G)|$ .

**Directed shallow minors.** The theory of directed minors is by far not as established as its undirected counterpart, in particular, there are several competing notions of directed minors.

► **Definition 2.1.** A *butterfly contraction* is the operation of contracting an edge  $e = (u, v)$  where either  $u$  has out-degree 1 or  $v$  has in-degree 1. A graph  $H$  is said to be a *butterfly minor* of a graph  $G$ , written  $H \preceq^b G$ , if it can be obtained from  $G$  by a series of vertex and edge deletions and butterfly contractions.

For undirected graphs, the notion of minors that are obtained by series of vertex and edge deletions and edge contractions can equivalently be defined in terms of minor models. In the directed setting these two notions are different (every butterfly minor is also a directed minor but not vice versa) [23].

► **Definition 2.2.** A digraph  $H$  has a *directed model* in a digraph  $G$  if there is a function  $\delta$  mapping vertices  $v \in V(H)$  of  $H$  to sub-graphs  $\delta(v) \subseteq G$  and edges  $e \in E(H)$  to edges  $\delta(e) \in E(G)$  such that

- if  $v \neq u$  then  $\delta(v) \cap \delta(u) = \emptyset$ ;
- 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  $\text{in}(\delta(v)) := V(\delta(v)) \cap \bigcup_{e=(u,v) \in E(H)} V(\delta(e))$  and  $\text{out}(\delta(v)) := V(\delta(v)) \cap \bigcup_{e=(v,w) \in E(H)} V(\delta(e))$ .

We furthermore require that for every  $v \in V(H)$

- there is a directed path in  $\delta(v)$  from any  $u \in \text{in}(\delta(v))$  to every  $u' \in \text{out}(\delta(v))$ ;
- there is at least one source vertex  $s_v \in \delta(v)$  that reaches every element of  $\text{out}(\delta(v))$ ;
- there is at least one sink vertex  $t_v \in \delta(v)$  that can be reached from every element of  $\text{in}(\delta(v))$ .

We write  $H \preceq^d G$  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.

► **Definition 2.3.** For  $r \geq 0$ , a digraph  $H$  is a *depth-r minor* of a digraph  $G$ , denoted as  $H \preceq_r^d G$ , 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 are bounded by  $r$ .

Finally, we consider the notion of directed topological minors.

► **Definition 2.4.** A digraph  $H$  is a *topological minor* of a digraph  $G$  if there is a function  $\delta$  mapping vertices  $v \in V(H)$  to vertices of  $V(G)$  and edges  $e \in E(H)$  to directed paths in  $G$  such that  $\delta(v) \neq \delta(u)$  for all distinct  $u, v \in V(H)$ , and 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  $\delta(e')$  with  $e' \in E(H)$ ,  $e' \neq e$ . For  $r \geq 0$ ,  $H$  is a *topological depth- $r$  minor* of  $G$ , written  $H \preceq_r^t G$ , if it is a topological minor and all paths  $\delta(e)$  have length at most  $2r$ .

► **Lemma 2.5.** For all digraphs  $H, G$  and  $r \geq 0$  it holds that  $H \preceq_r^t G$  implies  $H \preceq_r^d G$ .

The key to relating the edge density of depth- $r$  minors and depth- $r$  topological minors in the directed setting is based on a special requirement on directed bipartite graphs.

► **Definition 2.6.** A *directed bipartite graph* is a directed graph  $G = (A \dot{\cup} B, E)$  whose vertex set is partitioned into two sets  $A$  and  $B$  and  $E \subseteq A \times B$ .

The reason is that the branch sets of directed bipartite graphs can be chosen to have a particularly simple form.

► **Definition 2.7.** An *in-branching* is an orientation of a rooted tree with all edges oriented towards the root, an *out-branching* is defined analogously as a tree with all edges oriented away from the root.

► **Lemma 2.8** (see [23]). If  $H$  is a directed bipartite graph with  $H \preceq^d G$ , we can choose the branch-sets of the model of  $H$  in  $G$  to be in- or out-branchings. In this case  $H \preceq^d G \Leftrightarrow H \preceq^b G$ .

### 3 Classes of bounded expansion

Following [26] (see [23] for the directed case), we define classes of digraphs of bounded expansion by bounding the density of bounded depth minors.

► **Definition 3.1.** Let  $G$  be a digraph and let  $r \geq 0$ . The *greatest reduced average degree of rank  $r$*  (short *grad*) of  $G$ , denoted  $\nabla_r(G)$  is

$$\nabla_r(G) := \max \left\{ \frac{|E(H)|}{|V(H)|} : H \preceq_r^d G \right\}$$

and its *topological greatest average degree of rank  $r$*  (short *top-grad*) is

$$\tilde{\nabla}_r(G) := \max \left\{ \frac{|E(H)|}{|V(H)|} : H \preceq_r^t G \right\}.$$

► **Definition 3.2.** A class  $\mathcal{C}$  of digraphs has *bounded expansion* if there is a function  $f : \mathbb{N} \rightarrow \mathbb{N}$  such that for all  $r \geq 0$  it holds that  $\nabla_r(G) \leq f(r)$  for all  $G \in \mathcal{C}$ .

► **Example 3.3.** Every class of digraphs of bounded expansion has bounded edge density, hence not every class of acyclic digraphs has bounded expansion. The class of subdivided cliques with all edges oriented away from the subdivision vertices has bounded expansion. The only directed (topological) minors of a graph  $G$  from this class are the subgraphs of  $G$ , and hence it holds that  $\nabla_r(G) \leq 2$  for all  $r \geq 0$ . The underlying undirected class is not even nowhere dense in the undirected setting.

In the undirected case, we can give an equivalent definition of bounded expansion classes in terms of densities of topological depth- $r$  minors, due to the following theorem proved by Dvořák [10].

► **Theorem** (Theorem 3.9 of [10]). *Let  $r, d \geq 1$  and let  $p = 4(4d)^{(r+1)^2}$ . Let  $G$  be an undirected graph. If  $\nabla_r(G) \geq p$ , then  $\tilde{\nabla}_r(G) \geq d$ .*

A slightly worse bound can be given in the directed case, the only reason that we do not achieve the same bounds is that we may lose more edges when going to a bipartite subgraph than in the undirected case.

► **Lemma 3.4.** *Every digraph  $G$  contains a bipartite subgraph  $H$  with  $d(v) \geq \frac{1}{8}\nabla_0(G)$ , for all  $v \in V(H)$ .*

We can now follow the lines of Dvořák’s proof to obtain the following theorem.

► **Theorem 3.5.** *Let  $r, d \geq 1$  and let  $p = 32 \cdot (4d)^{(r+1)^2}$ . Let  $G$  be a digraph. If  $\nabla_r(G) \geq p$ , then  $\tilde{\nabla}_r(G) \geq d$ .*

► **Corollary 3.6.** *A class  $\mathcal{C}$  of digraphs has bounded expansion if and only if there is a function  $f: \mathbb{N} \rightarrow \mathbb{N}$  such that for all  $r \in \mathbb{N}$  it holds that  $\tilde{\nabla}_r(G) \leq f(r)$  for all  $G \in \mathcal{C}$ .*

## 4 Generalised colouring numbers

The *colouring number*  $\text{col}(G)$  of an undirected graph  $G$  is the minimum integer  $k$  such that there is a linear order  $<_L$  of the vertices of  $G$  for which each vertex  $v$  has *back-degree* at most  $k - 1$ , i.e. at most  $k - 1$  neighbours  $u$  with  $u <_L v$ . It is well-known that for any graph  $G$ , the chromatic number  $\chi(G)$  satisfies  $\chi(G) \leq \text{col}(G)$ .

Some generalisations of the colouring number of a graph have been studied in the literature. These include the *arrangeability* [6] used in the study of Ramsey numbers of graphs, the *admissibility* [19], and the *rank* [18] used in the study of the game chromatic number of graphs. Three natural generalisations of the colouring number are the series  $\text{adm}_r$ ,  $\text{col}_r$  and  $\text{wcol}_r$  of *generalised colouring numbers* introduced by Kierstead and Yang [20] (see Dvořák [11] for the general definition of  $\text{adm}_r$ ) in the context of colouring games and marking games on graphs. As proved by Zhu [38], these invariants can be used to characterise bounded expansion classes of graphs.

In this section, we define directed versions of the above invariants and show that also directed classes of bounded expansion can be characterised by bounds on the generalised colouring numbers.

► **Definition 4.1.** Let  $G$  be a digraph. By  $\Pi(G)$  we denote the set of all linear orders of  $V(G)$ . For  $L \in \Pi(G)$ , we write  $u <_L v$  if  $u$  is smaller than  $v$  with respect to  $L$ , and  $u \leq_L v$  if  $u <_L v$  or  $u = v$ . Let  $u, v \in V(G)$ . For a  $r \geq 0$ , we say that  $u$  is *weakly  $r$ -reachable* from  $v$  with respect to  $L$ , if there is a path  $P$  of length  $\ell$ ,  $0 \leq \ell \leq r$ , connecting  $u$  and  $v$  (in either direction) such that  $u$  is minimum among the vertices of  $P$  (with respect to  $L$ ). By  $\text{WReach}_r[G, L, v]$  we denote the set of vertices that are weakly  $r$ -reachable from  $v$  w.r.t.  $L$ .

Vertex  $u$  is *strongly  $r$ -reachable* from  $v$  with respect to  $L$ , if there is a path  $P$  of length  $\ell$ ,  $0 \leq \ell \leq r$ , connecting  $u$  and  $v$  (in either direction) such that  $u \leq_L v$  and such that all internal vertices  $w$  of  $P$  satisfy  $v <_L w$ . Let  $\text{SReach}_r[G, L, v]$  be the set of vertices that are strongly  $r$ -reachable from  $v$  w.r.t.  $L$ . Note that we have  $v \in \text{SReach}_r[G, L, v] \subseteq \text{WReach}_r[G, L, v]$ .

► **Definition 4.2.** For a non-negative integer  $r$ , we define the *weak  $r$ -colouring number*  $\text{wcol}_r(G)$  of  $G$  and the  *$r$ -colouring number*  $\text{col}_r(G)$  of  $G$  respectively as

$$\begin{aligned}\text{wcol}_r(G) &:= \min_{L \in \Pi(G)} \max_{v \in V(G)} |\text{WReach}_r[G, L, v]|, \\ \text{col}_r(G) &:= \min_{L \in \Pi(G)} \max_{v \in V(G)} |\text{SReach}_r[G, L, v]|.\end{aligned}$$

► **Definition 4.3.** For a non-negative integer  $r$ , the  *$r$ -admissibility*  $\text{adm}_r[G, L, v]$  of  $v$  w.r.t.  $L$  is the maximum size  $k$  of a family  $\{P_1, \dots, P_k\}$  of paths of length at most  $r$  with one end  $v$ , and the other end at a vertex  $w$  with  $w \leq_L v$ , and satisfy  $V(P_i) \cap V(P_j) = \{v\}$  for all  $1 \leq i < j \leq k$ . As for  $r > 0$  we can always let the paths end in the first vertex smaller than  $v$ , we can assume that the internal vertices of the paths are larger than  $v$ . Note that  $\text{adm}_r[G, L, v]$  is an integer, whereas  $\text{WReach}_r[G, L, v]$  and  $\text{SReach}_r[G, L, v]$  are vertex sets. The  *$r$ -admissibility*  $\text{adm}_r(G)$  of  $G$  is

$$\text{adm}_r(G) = \min_{L \in \Pi(G)} \max_{v \in V(G)} \text{adm}_r[G, L, v].$$

As in the undirected setting, we can show that these measures are strongly related.

► **Theorem 4.4.** *Let  $G$  be a digraph and let  $r \geq 1$ . Then*

$$\text{col}_r(G) \leq 2 \cdot (\text{adm}_r(G) - 1)^r + 1 \quad \text{and} \quad \text{wcol}_r(G) \leq 2 \cdot \text{adm}_r(G)^r.$$

In the following, we will prove that the above invariants can also be used to characterise bounded expansion classes of digraphs.

► **Definition 4.5.** Let  $G$  be a digraph,  $X \subseteq V(G)$ ,  $u \in V(G) \setminus X$  and  $r \in \mathbb{N}$ . The  *$r$ -projection* of  $u$  onto  $X$  is the set  $M_r^G(u, X)$  of all vertices  $v \in X$  such that there is a directed path between  $u$  and  $v$  in  $G$  (in either direction) of length at most  $r$  with all internal vertices in  $V(G) \setminus X$ .

The next lemma is proved as in the undirected case, compare e.g. to Lemma 2.9 of [9].

► **Lemma 4.6.** *Let  $G$  be a digraph,  $r \geq 0$  and  $X \subseteq V(G)$ . There exists a set  $\text{cl}_r(X) \subseteq V(G)$ , called an  *$r$ -closure* of  $X$  with the following properties. Let  $\xi := \lceil 2\nabla_{r-1}(G) \rceil$ .*

1.  $X \cap \text{cl}_r(X) = \emptyset$ ;
2.  $|\text{cl}_r(X)| \leq (r-1)\xi \cdot |X|$ ; and
3.  $|M_r^{G-\text{cl}_r(X)}(u, X)| \leq \xi$  for all  $u \in V(G) \setminus (X \cup \text{cl}_r(X))$ .

Just as in the undirected case, we know how to find an optimal order for the  $r$ -admissibility of a digraph (see [11], Algorithm 2, for a proof). For a set  $S \subseteq V(G)$  and  $v \in S$ , let  $b_r(S, v)$  be the maximum number of directed paths from  $v$  of length at most  $r$  intersecting only in  $v$  whose internal vertices belong to  $V(G) \setminus S$  and whose end-vertices belong to  $S$ .

► **Lemma 4.7** (see [11]). *Let  $G$  be a digraph and let  $L$  be the order of  $V(G)$  obtained iteratively as follows. Let  $S := V(G)$ . For  $i = n, n-1, \dots, 1$ , choose  $v_i \in S$  minimising  $p_i = b_r(S, v_i)$  and set  $S := S \setminus \{v_i\}$ . Then  $L$  is optimal for  $\text{adm}_r(G)$ .*

Clearly, the  $\text{adm}_r$ -value of the computed linear order is  $\max_{1 \leq i \leq n} p_i$ . Hence, according to the lemma, if  $\text{adm}_r(G) = c$ , then we can find a set  $S \subseteq V(G)$  such that every  $v \in S$  satisfies  $b_r(v, S) \geq c$ . Based on this obstruction for small admissibility, we are going to relate the grad of  $G$  to its admissibility (compare with Lemma 3.4 of [38] and Theorem 3.1 of [15] for the undirected case).

► **Theorem 4.8.** *For every digraph  $G$  and every  $r \in \mathbb{N}$  it holds that  $\text{adm}_r(G) < 16r^2 \nabla_{r-1}(G)^4$ .*

**Proof.** Let  $\xi := 2\nabla_{r-1}(G)$  and assume  $\text{adm}_r(G) \geq c := r^2\xi^4$ . According to Lemma 4.7, there exists a set  $S \subseteq V(G)$  such that every  $v \in S$  satisfies  $b_r(v, S) \geq c$ . We construct  $\text{cl}_r(S)$  according to Lemma 4.6, which has size at most  $(r-1)\xi \cdot |S|$ . We now iteratively contract short paths leading from  $S$  to  $\text{cl}_r(S)$ . Define  $\mathcal{P}_0$  as the set of paths between  $v \in S$  and  $w \in \text{cl}_r(S) \cup S$  of length at most  $r$  with all internal vertices in  $V(G) \setminus (\text{cl}_r(S) \cup S)$  (if two paths have the same initial and terminal vertex, we add only one of them). We hence have  $|\mathcal{P}_0| \geq \frac{c}{2} \cdot |S|$  by assumption.

As long as there exists  $P \in \mathcal{P}_i$ , contract  $P$  to an edge and remove from  $\mathcal{P}_i$  all paths which intersect  $P$  to obtain  $\mathcal{P}_{i+1}$ . As every internal vertex  $u$  of  $P$  satisfies  $|M_r^{G-\text{cl}_r(S)}(u, S)| \leq \xi$  by assumption,  $P$  can intersect with at most  $r\xi^2$  many other paths  $P' \in \mathcal{P}_i$ . Hence, hence after  $i+1$  contractions, we have  $|\mathcal{P}_{i+1}| \geq \frac{c}{2}|S| - (i+1)r\xi^2$ . Note that we are constructing a graph  $H \preceq_{r-1}^d G$  with vertex set  $S \cup \text{cl}_r(S)$ , that is, with at most  $((r-1)\xi + 1) \cdot |S|$  vertices, which by assumption on  $\nabla_{r-1}(G)$  can have at most  $\xi/2 \cdot ((r-1)\xi + 1) \cdot |S|$  many edges. This gives a contradiction for  $c > 2r\xi^2((r-1)\xi + 1) \cdot \xi/2$ , e.g. for  $c = r^2\xi^4 = 16r^2\nabla_{r-1}(G)^4$ . ◀

To complete the characterisation, we show the following.

► **Theorem 4.9.** *For every digraph  $G$  and every  $r \in \mathbb{N}$  it holds that  $\tilde{\nabla}_r(G) \leq 16(\text{adm}_{2r}(G) + 1)$ .*

**Proof.** Let  $c := \text{adm}_{2r}(G) + 1$ . Assume towards a contradiction that there is  $H \preceq_r^t G$  of edge density  $16c$ . Let  $H' \subseteq H$  be a bipartite graph with minimum degree at least  $2c$ , which exists by Lemma 3.4. Let  $L$  be an order of  $V(G)$  witnessing that  $\text{adm}_{2r}(G) = c$ . Let  $v$  be a principal vertex of  $H'$  and let  $\{P_1, \dots, P_t\}$  be the set of paths corresponding to the edges connecting  $v$  with its neighbours in  $H'$ . At most  $c$  paths among  $P_1, \dots, P_t$  contain an internal vertex that is smaller than  $v$  with respect to  $L$ , as otherwise,  $\text{adm}_{2r}(G) \geq c$ . Remove all edges  $e$  from  $H'$  that correspond to a path  $P_e$  in  $G$  such that the principal vertex is larger than some internal vertex of  $P_e$  to obtain a graph  $H''$ . By the above argument,  $H''$  has minimum degree at least  $c$ . Hence every vertex  $v$  reaches in  $G$  at least  $c$  vertices by paths that are internally vertex disjoint and contain no vertex smaller than  $c$ . Considering the largest vertex of  $H''$  in  $G$  with respect to  $L$ , this is a contradiction to our assumption. ◀

► **Corollary 4.10.** *A class  $\mathcal{C}$  of digraphs has bounded expansion if, and only if, there is a function  $f: \mathbb{N} \rightarrow \mathbb{N}$  such that  $\text{wcol}_r(G) \leq f(r)$  for all  $G \in \mathcal{C}$  and all  $r \geq 1$ .*

Finally, let us note that we can efficiently compute for every graph  $G$  from a bounded expansion class  $\mathcal{C}$  an order  $L$  of  $V(G)$  witnessing that the  $r$ -admissibility is small. The following lemma, which follows from a simple colour coding argument (see [7, Chapter 5.2 and 5.6]), shows that we can efficiently compute the number  $b_r(S, v)$  for every  $S \subseteq V(G)$  and every  $v \in V(G)$  if this number, or decide that the vertex will not be the next in the order.

► **Lemma 4.11.** *There is an algorithm which, given a digraph  $G$ , a set  $S \subseteq V(G)$ , a vertex  $v \in S$  and numbers  $r, k$ , decides whether there are  $k$  disjoint paths from  $v$  to vertices in  $S$  which are internally disjoint from  $S$  in time  $2^{k+r} \cdot n^{\mathcal{O}(1)}$ .*

► **Corollary 4.12.** *Let  $\mathcal{C}$  be a class of digraphs of bounded expansion. There is a function  $g$  such that for all  $r \geq 0$  and all  $G \in \mathcal{C}$  we can compute an optimal order for  $\text{adm}_r(G)$  in time  $g(r) \cdot n^{\mathcal{O}(1)}$ .*



## 5 Neighbourhood Complexity

Recently, a measure called *distance- $r$  neighbourhood complexity* was used to characterise classes of undirected bounded expansion [33]. Similar measures can be defined in the directed setting.

► **Definition 5.1.** Let  $G$  be a digraph, let  $X \subseteq V(G)$  and let  $r \geq 1$ . The *distance- $r$  out-neighbourhood complexity* of  $X$  in  $G$ , denoted  $\nu^+(G)$ , is defined by

$$\nu^+(G, X) = \max_{H \subseteq G, X \subseteq V(H)} |\{N_r^+(v) \cap X : v \in V(H)\}|.$$

Analogously, one can define the *distance- $r$  in-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.

Closure under subgraphs in the above definition is required to characterise sparse graph classes. Classically, this closure is not part of the definition, when it is e.g. used to define classes of bounded VC-dimension [35, 36, 37].

It was proved in [33] 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  $X \subseteq V(G)$  it holds that  $\nu(G, X) \leq c_r \cdot |X|$ . The analogous statement for classes of directed graphs does not hold, not even for  $r = 1$ , due to the simple fact that a directed bipartite graph does not contain directed minors other than its subgraphs.

► **Theorem 5.2.** *For every  $k \geq 1$  there exists a class  $\mathcal{C}_k$  of digraphs such that for all  $G \in \mathcal{C}_k$  and all  $r \geq 0$  it holds that  $\nabla_r(G) \leq k$  (hence  $\mathcal{C}_k$  has bounded expansion) and for each  $G \in \mathcal{C}_k$  there exists  $X \subseteq V(G)$  such that  $\nu_1^+(G, X) = |X|^k$ .*

For every bounded expansion class of digraphs we do obtain polynomial bounds though. To prove the following lemma, we use Lemma 4.6 to show that there are only few high degree vertices in the  $r$ -neighbourhood of  $X \subseteq V(G)$ .

► **Theorem 5.3.** *Let  $\mathcal{C}$  be a class of digraphs of bounded expansion. Then for all  $r \geq 1$  there exists  $k \geq 1$  such that for all  $G \in \mathcal{C}$  and  $X \subseteq V(G)$  we have  $\nu_r^+(G, X) \leq |X|^k$ . The same statement holds for in-neighbourhood complexity and mixed neighbourhood complexity.*

## 6 A Splitter Game for Classes of Digraphs of Bounded Expansion

In this section we establish a very useful property of bounded expansion classes of digraphs based on a directed version of a game, known as the *splitter game*, originally introduced as a characterisation of nowhere dense classes of undirected graphs in [16].

Let  $G$  be a digraph and let  $\ell, m, r \geq 0$ . The  $(\ell, m, r)$ -strong directed splitter game on  $G$  is played by two players, *Connector* and *Splitter*, as follows. Let  $G_0 := G$ . In round  $i + 1$  of the game, Connector picks a vertex  $v_{i+1} \in V(G_i)$ . Then Splitter chooses a subset  $W_{i+1} \subseteq V(G_i)$  with  $|W_{i+1}| \leq m$ . Define  $G_{i+1}$  as the induced subgraph of  $G_i$  with  $V(G_{i+1}) = \tilde{N}_{G_i, r}(v_{i+1}) \setminus W_{i+1}$ . Splitter wins if  $V(G_{i+1}) = \emptyset$ . Otherwise the game continues to the next round. If Splitter has not won after  $\ell$  rounds, then Connector wins.

A *strategy* for Splitter is a function  $f$  associating to every partial play  $(v_1, W_1, \dots, v_s, W_s)$  with associated sequence  $G_0, \dots, G_s$  and every move  $v_{s+1} \in A_s$  by Connector a move  $W_{s+1} \subseteq V(G_s)$  with  $|W_{s+1}| \leq m$  for Splitter. A strategy  $f$  is a *winning strategy* for Splitter if she wins every play in which she follows the strategy  $f$ . If such a winning strategy exists, we say that Splitter *wins* the  $(\ell, m, r)$ -directed splitter game on  $G$ .

For undirected graphs the splitter game can be used to characterise nowhere dense classes of graphs. This is not the case for directed graphs, however, short winning strategies can be provided for bounded expansion classes (compare to [22]).

► **Theorem 6.1.** *Let  $G$  be a graph, let  $r \in \mathbb{N}$  and let  $\ell = \text{wcol}_{4r}(G)$ . Then splitter wins the  $(\ell, 1, r)$ -strong splitter game.*

**Proof.** Let  $L$  be a linear order that witnesses  $\text{wcol}_{4r}(G) = \ell$ . First note the following. Let  $v \in V(G)$  and let  $m \in \tilde{N}_r(v)$  be the  $L$ -minimal element of  $\tilde{N}_r(v)$ . Then for every  $w \in \tilde{N}_r(v)$ ,  $G[\tilde{N}_r(v)]$  contains a directed path from  $w$  to  $m$  of length at most  $4r$ . Hence,  $m \in \text{WReach}(G, L, w)$  for every  $w \in \tilde{N}_r(v)$ .

We now describe a winning strategy for splitter in the  $(\ell, 1, r)$ -splitter game. Suppose in round  $i + 1 \leq \ell$ , connector chooses a vertex  $v_{i+1} \in V(G_i)$ . Let  $W_{i+1}$  (splitter's choice) be the minimum vertex of  $\tilde{N}_{G_i, r}(v_{i+1})$  with respect to  $L$ . Then for each  $u \in N_{G_i, r}(v_{i+1})$  there is a path between  $u$  and  $w_{i+1}$  of length at most  $4r$  that uses only vertices of  $N_r^{G_i}(v_{i+1})$ . As  $w_i$  is  $L$ -minimal in  $N_r^{G_i}(v_{i+1})$ ,  $w_{i+1}$  is weakly  $4r$ -reachable from each  $u \in N_r^{G_i}(v_{i+1})$ . Now let  $G_{i+1} := G_i[N_r^{G_i}(v_{i+1}) \setminus \{w_{i+1}\}]$ . As  $w_{i+1}$  is not part of  $G_{i+1}$ , in the next round splitter will choose another vertex which is weakly  $4r$ -reachable from every vertex of the remaining  $r$ -neighbourhood. As  $\text{wcol}_{4r}(G) = \ell$ , the game must stop after at most  $\ell$  rounds. ◀

Note that unlike the undirected case of nowhere dense classes of graphs, the strong splitter game is not a characterisation of bounded expansion classes, as splitter wins the  $(1, 1, 1)$ -strong splitter game on every acyclic digraph, but the class of acyclic digraphs does not have bounded expansion.

## 7 Neighbourhood Covers

Neighbourhood covers of small radius and small size play a key role in the design of many data structures for distributed systems. There is also a deep connection between sparse neighbourhood covers of small radius and sparse graph spanners of low stretch. In this section we will show that classes of digraphs of bounded expansion admit sparse strong neighbourhood covers that can be computed by a fixed-parameter algorithm.

► **Definition 7.1.** Let  $r \in \mathbb{N}$ . A *strong  $r$ -neighbourhood cover*  $\mathcal{X}$  of a graph  $G$  is a mapping  $\mathcal{X} : V(G) \rightarrow 2^{V(G)}$  such that  $G[\mathcal{X}(v)]$  is strongly connected and  $\tilde{N}_r(v) \subseteq \mathcal{X}(v)$ . We call each  $G[\mathcal{X}(v)]$  a *cluster* of  $\mathcal{X}$ .

The *radius* of a cluster  $C := G[\mathcal{X}(v)]$  is defined as the minimal  $r \in \mathbb{N}$  for which there is a vertex  $w \in V(C)$  and for every  $w \in V(C)$ , the cluster  $C$  contains a directed path of length at most  $r$  from  $w$  to  $v$  and a directed path of length at most  $r$  from  $v$  to  $w$ . The *radius*  $\text{rad}(\mathcal{X})$  of a cover  $\mathcal{X}$  is the maximum radius of any of its clusters.

The *degree*  $d^{\mathcal{X}}(v)$  of  $v$  in  $\mathcal{X}$  is the number of clusters that contain  $v$ . The *maximum degree*  $\Delta(\mathcal{X})$  of  $\mathcal{X}$  is  $\Delta(\mathcal{X}) = \max_{v \in V(G)} d^{\mathcal{X}}(v)$ .

The main result of this section is the following theorem.

► **Theorem 7.2.** *Let  $\mathcal{C}$  be a class of digraphs of bounded expansion. There are functions  $f, h : \mathbb{N} \rightarrow \mathbb{N}$  such that for all  $r \in \mathbb{N}$  and all graphs  $G \in \mathcal{C}$ , there exists a strong  $r$ -neighbourhood cover of radius at most  $4r$  and maximum degree at most  $f(r)$  and this cover can be computed in time  $h(r) \cdot n^{\mathcal{O}(1)}$ .*

In the next lemma we use the weak colouring number to prove the existence of sparse neighbourhood covers in bounded expansion classes of digraphs.

► **Definition 7.3.** Let  $G$  be a digraph, let  $<$  be an ordering of  $V(G)$  and let  $r > 0$ . For a vertex  $v \in V(G)$  we define  $X_r[G, <, v]$  as the set of vertices  $w \in V(G)$  such that  $v \in \text{WReach}_r[G, <, w]$ .

► **Lemma 7.4.** Let  $G$  be a graph such that  $\text{wcol}_{4r}(G) \leq s$  and let  $<$  be an order witnessing this. For  $v \in V(G)$ , let  $m(v)$  be the minimum of  $\tilde{N}_r(v)$  with respect to  $<$ . Then  $\mathcal{X} : V(G) \rightarrow 2^{V(G)}$  with  $\mathcal{X}(v) = X_{4r}[G, <, m(v)]$  is a strong  $r$ -neighbourhood cover of  $G$  with radius at most  $4r$  and maximum degree at most  $s$ .

**Proof.** Clearly, by construction of the sets  $X_{4r}[G, <, v]$  the radius of each cluster is at most  $4r$ . Furthermore, for  $v \in V(G)$  we have  $\tilde{N}_r(v) \subseteq \mathcal{X}(v)$ . To see this, let  $m(v)$  be the minimum of  $\tilde{N}_r(v)$  with respect to  $<$ . Then  $m(v)$  is weakly  $4r$ -reachable from every  $w \in N_r(v) \setminus \{m(v)\}$ . There is a path from  $w$  to  $v$  of length at most  $2r$  (on the closed walk containing both  $w$  and  $v$ ) and a path from  $v$  to  $m(v)$  of length at most  $2r$  (again on a closed walk containing the two). Now the concatenation of the two paths is a walk of length at most  $4r$  which contains as a sub-walk which is a path of length at most  $4r$ . As this path uses only vertices of  $\tilde{N}_r(v)$  and  $m(v)$  is the minimum element, we conclude that  $\tilde{N}_r(v) \subseteq X_{4r}[G, <, m(v)]$ . Finally observe that for every  $v \in V(G)$ ,

$$\begin{aligned} d^{\mathcal{X}}(v) &= |\{u \in V(G) : v \in X_{4r}[G, <, u]\}| \\ &= |\{u \in V(G) : u \in \text{WReach}_{4r}[G, <, v]\}| = |\text{WReach}_{4r}[G, <, v]| \leq s. \end{aligned} \quad \blacktriangleleft$$

Now to prove Theorem 7.2 it suffices to note that the clusters  $X_{4r}[G, <, v]$  can be computed in the desired time. According to Corollary 4.12, we can compute an order  $<$  which is optimal for  $\text{adm}_{4r}(G)$  in time  $g(4r) \cdot n^{\mathcal{O}(1)}$ . According to Theorem 4.4, this order also satisfies  $|\text{WReach}_{4r}[G, <, v]| \leq f(r)$  for properly defined function  $f$ . We order the vertices of  $G$  in order  $<$ . Now, to compute  $X_{4r}[G, <, v]$  for a vertex  $v$ , we just have to perform the first  $4r$  levels of a breadth-first search around  $v$  (where we follow paths either in the direction of the edges or against the direction of the edges) which stops in every branch if a vertex smaller than  $v$  is encountered. Defining  $g$  accordingly finishes the proof of the theorem.

## 8 Constant-Factor Approximation Algorithms for Strong Dominating Sets

In this section we prove that strong dominating sets can be approximated up to a constant factor on any class  $\mathcal{C}$  of directed bounded expansion. Our approach is inspired by [11].

► **Definition 8.1** (Strong  $r$ -Dominating Sets).

1. Let  $r \geq 1$  and let  $G$  be a digraph. A vertex  $v \in V(G)$  *strongly- $r$ -dominates* a vertex  $u \in V(G)$  if there is a closed walk of length at most  $2r$  in  $G$  containing  $u$  and  $v$ .
2. A *strong- $r$ -dominating set* is a set  $X \subseteq V(G)$  such that every vertex in  $G$  is strongly dominated by a vertex in  $X$ .
3. The *strong  $r$ -domination number* of  $G$ , denoted  $\text{sdom}_r(G)$ , is the minimum size of a strong  $r$ -dominating set of  $G$ .

► **Theorem 8.2.** Let  $\mathcal{C}$  be a class of digraphs of directed bounded expansion. Let  $r \geq 1$ . There is a polynomial time constant factor approximation algorithm for strong  $r$ -dominating sets. More precisely, for every value of  $r$ , there is an algorithm running in time  $g(r) \cdot n^{\mathcal{O}(1)}$  for some function  $g$  which, on input  $G \in \mathcal{C}$  computes a strong- $r$ -dominating set  $D \subseteq V(G)$  of order at most  $\text{wcol}_{4r}(G)^2 \cdot \text{sdom}_r(G)$ .

In the remainder of this section we prove Theorem 8.2. Let  $r$  be given. An  $r$ -obstruction set is a set  $X \subseteq V(G)$  such that for any distinct  $x, y \in X$ , there are *no* two closed directed walks  $W_1, W_2 \subseteq V(G)$ , each of length at most  $2r$ , such that  $W_1 \cap W_2 \neq \emptyset$  and  $x \in W_1$  and  $y \in W_2$ . Note that we do not require  $W_1 \neq W_2$ . We call a pair  $u, v$  of vertices which form an  $r$ -obstruction set  $\{u, v\}$   $r$ -separated. Otherwise, we call the pair  $u, v$   $r$ -dependent.

For a set  $X \subseteq V(G)$ , we define  $\text{Sdom}_r(X)$  as the set of vertices  $v \in V(G)$  such that  $v$  is strongly- $r$ -dominated by a vertex in  $X$ .

As no two distinct vertices of an obstruction set lie on a closed walk of length at most  $2r$ , no two vertices from the set can be dominated strongly  $r$ -dominated by a single vertex. Hence, if  $G$  contains an obstruction set of order  $k$  then  $\text{sdom}_r(G) \geq k$ .

► **Lemma 8.3.** *There exists a polynomial time algorithm which, given a number  $r \geq 1$  and a digraph  $G$  together with an ordering  $L$  witnessing  $\text{wcol}_{4r}(G)$ , computes an obstruction set  $X \subseteq V(G)$  of order  $k$ , for some  $k$ , and an  $r$ -dominating set of order at most  $\text{wcol}_{4r}(G)^2 \cdot k$ .*

**Proof.** Let  $L$  be an ordering of  $G$  witnessing  $\text{wcol}_{4r}(G)$ . We greedily compute sets  $A, D, S \subseteq V(G)$  as follows. Start with  $A_0 = D_0 := \emptyset$  and  $S_0 := V(G)$ . Now suppose  $A_i, D_i, S_i$  have already been defined. If  $S_i = \emptyset$  then the construction stops here. Otherwise, let  $a$  be the  $<_L$ -minimal element of  $S_i$  and define  $A_{i+1} := A_i \cup \{a\}$ ,  $D_{i+1} := D_i \cup \text{WReach}_{4r}(G, L, a)$ . Finally, we define  $S_{i+1} := S_i \setminus \text{Sdom}_r(\text{WReach}_{4r}(G, L, a))$ .

Now let  $i$  be minimal such that  $S_i = \emptyset$ . Such an index  $i$  exists as we add a vertex to  $A$  – and remove it from  $S$  – at each iteration.

Clearly,  $D := D_i$  is a strong  $r$ -dominating set of  $G$  and  $|D| \leq \text{wcol}_{4r}(G) \cdot |A|$ . We will show next that  $G$  contains an  $r$ -obstruction set  $X \subseteq A$  of order  $\frac{1}{\text{wcol}_{4r}(G)} \cdot |A|$ . Hence,  $D$  is a factor  $(\text{wcol}_{4r}(G))^2$  approximation of the strong  $r$ -domination number of  $G$ .

We construct a digraph  $H$  with vertex set  $A$  and edges  $(u, v)$  if  $u <_L v$  and  $u$  and  $v$  are  $r$ -dependent. We will show next that the maximal out-degree of  $H$  is  $< \text{wcol}_{4r}(G)$ . For  $a \in A$  let  $a_1, \dots, a_l$  be the elements of  $A$  which are  $L$ -smaller than  $a$  and such that  $a$  and  $a_i$  are  $r$ -dependent, for all  $1 \leq i \leq l$ . We claim that  $l < \text{wcol}_{4r}(G)$ .

For every  $1 \leq i \leq l$  let  $W_1 := W_1(i)$  and  $W_2 := W_2(i)$  be two closed directed walks in  $G$  of length at most  $2r$  which intersect each other and such that  $a \in V(W_1)$  and  $a_i \in V(W_2)$ . Such walks exist as  $a_i$  and  $a$  are  $r$ -dependent. Let  $z = z(i)$  be the  $L$ -minimal vertex in  $V(W_1 \cup W_2)$ . Note that  $z \in \text{WReach}_{4r}(G, L, a)$ . For,  $W_1 \cup W_2$  form a strongly connected subgraph on at most  $4r$  vertices containing  $z$  and  $a$  and hence there is a directed path from  $a$  to  $z$  of length at most  $4r$ . As  $z$  is the  $L$ -minimal element of  $W_1 \cup W_2$ , this implies that  $z \in \text{WReach}_{4r}(G, L, a)$ .

We claim first that  $z \notin V(W_1)$ . For otherwise,  $z$  would strongly  $r$ -dominate  $a$  and hence in the  $i$ -th iteration of the algorithm,  $a$  would have been removed from  $S_i$ , contradicting the fact that  $a \in A$ . Thus,  $z(i) \in W_2(i)$  for all  $1 \leq i \leq l$ .

Now suppose  $z(i) = z(j)$  for some  $1 \leq i < j \leq l$ . But then again  $a_j$ , which is contained in  $W_2(j)$ , is strongly  $r$ -dominated by  $z(i)$  and hence  $a_j$  would have been removed from  $S_i$  at step  $i$ . Hence,  $z(i) \neq z(j)$  for all  $1 \leq i \neq j \leq l$ . It follows that  $l \leq |\text{WReach}_{4r}(G, L, a)| - 1 < \text{wcol}_{4r}(G)$ .

This shows that the maximum outdegree of any vertex in  $H$  is  $< \text{wcol}_{4r}(G)$ . Hence,  $H$  is  $\text{wcol}_{4r}(G) - 1$ -degenerate and therefore  $\text{wcol}_{4r}(G)$ -colourable. Thus, the colour class  $C$  of maximal size contains at least  $\frac{1}{\text{wcol}_{4r}(G)} \cdot |A|$  elements of  $A$  and forms an  $r$ -obstruction set, witnessing that the strong  $r$ -domination number of  $G$  is at least  $\frac{1}{\text{wcol}_{4r}(G)^2} |A|$ . This completes the proof of the lemma. ◀

Theorem 8.2 now follows immediately from the previous lemma together with Corollary 4.12. Note, however, that our algorithm runs in polynomial time for any fixed  $r$  but it depends exponentially on  $\text{wcol}_{4r}(\mathcal{C})$ . The reason is that we currently do not know how to compute a good approximation of the  $\text{wcol}$ -ordering in polynomial time.

---

## References

- 1 Saeed Akhondian Amiri, Lukasz Kaiser, Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz. Graph searching games and width measures for directed graphs. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, pages 34–47, 2015.
- 2 Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.
- 3 Dietmar Berwanger, Anuj Dawar, Paul Hunter, and Stephan Kreutzer. Dag-width and parity games. In *Symp. on Theoretical Aspects of Computer Science (STACS)*, 2006.
- 4 Dietmar Berwanger, Anuj Dawar, Paul Hunter, Stephan Kreutzer, and Jan Obdržálek. The dag-width of directed graphs. *J. Comb. Theory, Ser. B*, 102(4):900–923, 2012. doi:10.1016/j.jctb.2012.04.004.
- 5 Dietmar Berwanger, Erich Grädel, Lukasz Kaiser, and Roman Rabinovich. Entanglement and the complexity of directed graphs. *Theor. Comput. Sci.*, 463:2–25, 2012. doi:10.1016/j.tcs.2012.07.010.
- 6 Guantao Chen and Richard H. Schelp. Graphs with linearly bounded ramsey numbers. *J. Combin. Theory Ser. B*, 57:138–149, 1993.
- 7 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 8 Anuj Dawar and Stephan Kreutzer. Domination problems in nowhere-dense classes of graphs. In *Foundations of software technology and theoretical computer science (FSTTCS)*, pages 157–168, 2009.
- 9 Pål Grønås Drange, Markus S. Dregi, Fedor V Fomin, Stephan Kreutzer, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, Felix Reidl, Saket Saurabh, Fernando Sánchez Villaamil, Sebastian Siebertz, et al. Kernelization and sparseness: the case of dominating set. *arXiv preprint arXiv:1411.4575*, 2014.
- 10 Zdenek Dvorák. Asymptotical structure of combinatorial objects. *Charles University, Faculty of Mathematics and Physics*, 51:357–422, 2007.
- 11 Zdeněk Dvořák. Constant-factor approximation of the domination number in sparse graphs. *European Journal of Combinatorics*, 34(5):833–840, 2013.
- 12 Zdenek Dvorak, Daniel Král, and Robin Thomas. Testing first-order properties for subclasses of sparse graphs. *J. ACM*, 60(5):36, 2013.
- 13 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.
- 14 Robert Ganian, Petr Hliněný, Joachim Kneis, Daniel Meister, Jan Obdržálek, Peter Rossmanith, and Somnath Sikdar. Are there any good digraph width measures? In Venkatesh Raman and Saket Saurabh, editors, *Parameterized and Exact Computation*, volume 6478 of *Lecture Notes in Computer Science*, pages 135–146. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-17493-3\_14.
- 15 Martin Grohe, Stephan Kreutzer, Roman Rabinovich, Sebastian Siebertz, and Konstantinos Stavropoulos. Colouring and covering nowhere dense graphs. In *Graph-Theoretic Concepts in Computer Science – 41st International Workshop, WG 2015, Garching, Germany, June 17-19, 2015, Revised Papers*, pages 325–338, 2015.

- 16 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.
- 17 Thor Johnson, Neil Robertson, Paul D. Seymour, and Robin Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001.
- 18 Hal A. Kierstead. A simple competitive graph coloring algorithm. *J. Combin. Theory Ser. B*, 78:57–68, 2000.
- 19 Hal A. Kierstead and William T. Trotter. Planar graph coloring with an uncooperative partner. In W.T. Trotter, editor, *Planar Graphs*, volume 9 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 85–93. AMS, 1993.
- 20 Hal A. Kierstead and Daqing Yang. Orderings on graphs and game coloring number. *Order*, 20:255–264, 2003.
- 21 Stephan Kreutzer and Sebastian Ordyniak. Digraph decompositions and monotonicity in digraph searching. In *Graph-Theoretic Concepts in Computer Science, 34th International Workshop, WG 2008, Durham, UK, June 30 – July 2, 2008. Revised Papers*, pages 336–347, 2008.
- 22 Stephan Kreutzer, Michał Pilipczuk, Roman Rabinovich, and Sebastian Siebertz. The generalised colouring numbers on classes of bounded expansion. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 – Kraków, Poland*, pages 85:1–85:13, 2016.
- 23 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.
- 24 Michael Lampis, Georgia Kaouri, and Valia Mitsou. On the algorithmic effectiveness of digraph decompositions and complexity measures. *Discrete Optimization*, 8(1):129–138, 2011.
- 25 Daniel Meister, Jan Arne Telle, and Martin Vatshelle. Characterization and recognition of digraphs of bounded kelly-width. In *Workshop on Graph Theoretical Aspects of Computer Science (WG)*, pages 270–279, 2007. doi:10.1007/978-3-540-74839-7\_26.
- 26 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion I. decompositions. *European Journal of Combinatorics*, 29(3):760–776, 2008.
- 27 Jaroslav Nešetřil and Patrice Ossona de Mendez. On nowhere dense graphs. *European Journal of Combinatorics*, 32(4):600–617, 2011.
- 28 Jaroslav Nešetřil and Patrice Ossona De Mendez. On low tree-depth decompositions. *Graphs and Combinatorics*, 31(6):1941–1963, 2015.
- 29 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion I–III. *European Journal of Combinatorics*, 29, 2008. Series of 3 papers appearing in volumes (3) and (4).
- 30 Jan Obdržálek. Dag-width: connectivity measure for directed graphs. In *Symp. on Discrete Algorithms (SODA)*, pages 814–821, 2006.
- 31 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*, page 475–484, 1997.
- 32 Felix Reidl. *Structural sparseness and complex networks*. Dr., Aachen, Techn. Hochsch., Aachen, 2016.
- 33 Felix Reidl, Fernando Sánchez Villaamil, and Konstantinos Stavropoulos. Characterising bounded expansion by neighbourhood complexity. *arXiv preprint arXiv:1603.09532*, 2016.
- 34 Mohammad Ali Safari. D-width: A more natural measure for directed tree width. In *Proc. of Mathematical Foundations of Computer Science (MFCS)*, number 3618 in Lecture Notes in Computer Science, pages 745–756, 2005.

- 35 Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, 1972.
- 36 Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1):247–261, 1972.
- 37 Vladimir N. Vapnik and Alexey Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of Complexity*, pages 11–30. Springer, 2015.
- 38 Xuding Zhu. Colouring graphs with bounded generalized colouring number. *Discrete Mathematics*, 309(18):5562–5568, 2009.





# Computing Majority by Constant Depth Majority Circuits with Low Fan-in Gates<sup>\*†</sup>

Alexander S. Kulikov<sup>1</sup> and Vladimir V. Podolskii<sup>2</sup>

- 1 Steklov Mathematical Institute, Russian Academy of Sciences, St. Petersburg, Russia  
kulikov@pdmi.ras.ru
- 2 Steklov Mathematical Institute, Russian Academy of Sciences, St. Petersburg, Russia; and  
National Research University, Higher School of Economics, St. Petersburg, Russia  
podolskii@mi.ras.ru

---

## Abstract

We study the following computational problem: for which values of  $k$ , the majority of  $n$  bits  $\text{MAJ}_n$  can be computed with a depth two formula whose each gate computes a majority function of at most  $k$  bits? The corresponding computational model is denoted by  $\text{MAJ}_k \circ \text{MAJ}_k$ . We observe that the minimum value of  $k$  for which there exists a  $\text{MAJ}_k \circ \text{MAJ}_k$  circuit that has high correlation with the majority of  $n$  bits is equal to  $\Theta(n^{1/2})$ . We then show that for a randomized  $\text{MAJ}_k \circ \text{MAJ}_k$  circuit computing the majority of  $n$  input bits with high probability for every input, the minimum value of  $k$  is equal to  $n^{2/3+o(1)}$ . We show a worst case lower bound: if a  $\text{MAJ}_k \circ \text{MAJ}_k$  circuit computes the majority of  $n$  bits correctly on all inputs, then  $k \geq n^{13/19+o(1)}$ . This lower bound exceeds the optimal value for randomized circuits and thus is unreachable for pure randomized techniques. For depth 3 circuits we show that a circuit with  $k = O(n^{2/3})$  can compute  $\text{MAJ}_n$  correctly on all inputs.

**1998 ACM Subject Classification** F.1.3 Complexity Measures and Classes

**Keywords and phrases** circuit complexity, computational complexity, threshold, majority, lower bound, upper bound

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.49

## 1 Introduction

In this paper we study majority functions and circuits consisting of them. These functions and circuits arise for various reasons in many areas of Computational Complexity (see e.g. [12, 14, 7]). In particular, the iterated majority function (or recursive majority) consisting of iterated application of majority of small number of variables to itself, turns out to be of great importance, helps in various constructions and provides an example of the function with interesting complexity properties in various models [8, 11, 13, 9].

One of the most prominent examples to illustrate this is the proof by Valiant [17] that the majority  $\text{MAJ}_n$  of  $n$  variables can be computed by a boolean circuit of depth  $5.3 \log n$ .

---

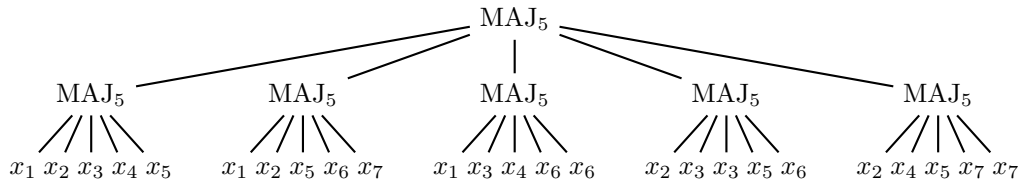
\* The full version of this paper is available as ECCC report TR16-158 – <https://eccc.weizmann.ac.il/report/2016/158/>.

† The research presented in Section 4 was supported by Russian Science Foundation (project 16-11-10123). The research presented in Section 5 was partially supported by grant MK-7312.2016.1 and by the Russian Academic Excellence Project '5-100'.



The construction of Valiant is randomized and there is no deterministic construction known achieving the same (or even reasonably close) depth parameter. The construction works as follows. Consider a uniform boolean formula (that is, tree-like circuit) consisting of  $5.3 \log n$  interchanging layers of AND and OR gates of fan-in 2. For each input to the circuit substitute a random variable of the function  $\text{MAJ}_n$ . Valiant showed that this circuit computes  $\text{MAJ}_n$  with positive probability. Note that AND and OR gates are precisely  $\text{MAJ}_2$  functions with different threshold values. Thus this construction can be viewed as a computation of  $\text{MAJ}_n$  by a circuit consisting of  $\text{MAJ}_2$  gates. There are versions of this construction with the circuits consisting of  $\text{MAJ}_3$  gates (see, e.g., [5]).

In this paper we study what happens with this setting if we restrict the depth of the circuit to a small constant. That is, we study for which  $k$  the function  $\text{MAJ}_n$  can be computed by small depth circuit consisting of  $\text{MAJ}_k$  gates. We mostly concentrate on depth 2 and denote the corresponding model by  $\text{MAJ}_k \circ \text{MAJ}_k$ . For example, the majority of  $n = 7$  bits  $x_1, x_2, \dots, x_7$  can be computed with the following  $\text{MAJ}_5 \circ \text{MAJ}_5$  circuit for  $k = 5$ :



We study which upper and lower bounds on  $k$  can be shown.

More context to the problem under consideration comes from the studies of boolean circuits of constant depth. The class  $\widehat{\text{TC}}^0$  of boolean functions computable by polynomial size constant depth circuits consisting of MAJ gates plays one of the central roles in this area. Its natural generalization is the class  $\text{TC}^0$  in which instead of MAJ gates one can use arbitrary linear threshold gates, that is analogs of the majorities in which variables are summed up with arbitrary integer coefficients and are compared with arbitrary integer threshold. It is known that to express any threshold function it is enough to use exponential size coefficients. To show that  $\text{TC}^0$  is actually the same class as  $\widehat{\text{TC}}^0$  it is enough to show that any linear threshold function can be computed by constant depth circuit consisting of threshold functions with polynomial-size coefficients (polynomial size can be simulated in  $\widehat{\text{TC}}^0$  by repetition of variables). It was shown by Siu and Bruck in [16] that any linear threshold function can be computed by polynomial size depth-3 majority circuit. This result was improved to depth-2 by Goldman, Håstad and Razborov in [4]. More generally, it was shown in [4] that depth- $d$  polynomial size threshold circuit can be computed by depth- $(d + 1)$  polynomial size majority circuit, in particular establishing the class of depth-2 threshold circuits as one of the weakest classes for which we currently do not know superpolynomial size lower bounds. The best lower bound known so far is  $\Omega(\frac{n^{3/2}}{\log^3 n})$  by Kane and Williams [10].

Note, however, that the result of [4] does not translate to monotone setting. Hofmeister in [6] showed that there is a monotone linear threshold function requiring exponential size depth-2 monotone majority circuit. Recently this result was extended by Chen, Oliveira and Servedio [2] to monotone majority circuits of arbitrary constant depth.

Our setting can be viewed as a scale down of the setting of [4] and [6]. In [4, 6] exponential weight threshold functions are compared to depth-2 threshold circuits with polynomial weights. In our setting we compare weight- $n$  threshold functions with depth-2 threshold circuits with weights  $k$ . In this paper we consider monotone setting.

Another context to our studies comes from the studies of lower bounds against  $\widehat{\text{TC}}^0$ . Allender and Koucký in [1] showed that to prove that some function is not in  $\widehat{\text{TC}}^0$  it is enough

to show that some self-reducible function requires circuit-size at least  $n^{1+\varepsilon}$  when computed by constant depth majority circuit. As an intermediate result they show that  $\text{MAJ}_n$  can be computed by  $O(1)$ -depth circuit consisting of  $\text{MAJ}_{n^\varepsilon}$  gates and of size  $O(n \log n)$ . This setting is similar to ours, however in this paper we are interested in the precise depth and we do not pose additional bounds on the size of the circuit (however note that the bound on the fan-in  $k$  of the gates and the bound on the depth  $d$  of the circuit naturally imply the bound of  $O(k^d)$  on the size of the circuit).

We consider three models of computation of the majority function: computation on most of the inputs (that is, high correlation with the function), randomized computation with small error probability on all inputs, and deterministic computation with no errors. We prove the following lower and upper bounds for our setting.

**Circuits with high correlation.** We observe that the minimum value of  $k$  for which there exists a  $\text{MAJ}_k \circ \text{MAJ}_k$  circuit that computes  $\text{MAJ}_n$  correctly on  $2/3$  fraction of all the inputs, is equal to  $\Theta(n^{1/2})$ . A lower bound is proved by observing that a circuit with  $k = \alpha n^{1/2}$  does not even have a possibility to read a large fraction of input bits when the constant  $\alpha$  is small enough. We show that in this case the circuit errs on many inputs. An upper bound is proved for the following natural circuit: pick  $k = \Theta(n^{1/2})$  random subsets of the  $n$  inputs bits of size  $k$ , compute the majority for each of them, and then compute the majority of results. Such a circuit computes  $\text{MAJ}_n$  correctly with high probability on inputs whose weight is not too close to  $n/2$ . By tuning the parameters appropriately, we ensure that the middle layers of the boolean hypercube (containing inputs where the circuits errs with high probability) constitute only a small fraction of all the inputs.

**Randomized circuits.** We prove that for a probabilistic distribution  $\mathcal{C}$  of  $\text{MAJ}_k \circ \text{MAJ}_k$  circuits with a property that for every input  $A \in \{0, 1\}^n$  the probability that  $\mathcal{C}(A) = \text{MAJ}_n(A)$  is  $1 - \varepsilon$  for a constant  $\varepsilon > 0$ , the minimum value of  $k$  is  $n^{2/3}$ , up to polylogarithmic factors. A lower bound is proved by showing that a small circuit must err on a large fraction of minterms/maxterms of  $\text{MAJ}_n$ . Roughly, the majority function have many inputs  $A \in \{0, 1\}^n$  with a property that changing a single bit in  $A$  changes the value of the function (these are precisely minterms and maxterms of  $\text{MAJ}_n$ ). If  $k$  is small enough, a  $\text{MAJ}_k \circ \text{MAJ}_k$  circuit can reflect such a change in the value only for a small fraction of inputs. To show an upper bound, we split the  $n$  input bits into blocks and for each block compute several middle layers values of the bits of this block in sorted order. We then compute the majority of all the resulting values. We show that by tuning the parameters appropriately, one can ensure that this circuit err only on a polynomially small fraction of inputs.

**Deterministic circuits.** The trivial upper bound on  $k$  is  $k \leq n$ . We do not have any nontrivial upper bound on  $k$  for depth 2 circuits. We however have examples for  $n = 7, 9, 11$  of circuits with  $k = n - 2$ . For depth 3 we have an upper bound  $O(n^{2/3})$  which coincides with the optimal value for depth 2 randomized circuits up to polylogarithmic factor. We prove this upper bound by extending the construction of upper bound for depth 2 randomized circuits. We use an extra layer of the circuit to preorder the inputs. Regarding the lower bound for depth 2 we observe that the following simple special case cannot compute  $\text{MAJ}_n$ : each gate is a standard majority (that is, with threshold  $k/2$ ) of exactly  $k = n - 2$  distinct variables. Next, we proceed to the main result of the paper. We show that the minimum value of  $k$  for which there is a depth 2 circuit computing  $\text{MAJ}_n$  on all inputs is at least  $n^{13/19}$  up to a polylogarithmic factor.

Note that this lower bound exceeds the optimal value of  $k$  for randomized circuits. Thus, despite the fact that randomized techniques is extensively used for studying majority and circuits constructed from it and proves to be very powerful (recall for example Valiant's result [17]), in our setting using combinatorial methods we prove a lower bound that is unreachable for a pure probabilistic approach. The proof of this result however is still probabilistic: in essence we consider a circuit with  $k$  smaller than  $n^{13/19}$  and build a distribution on inputs that fools this circuit. The catch is that the distribution is tailored to fool this particular circuit: it is constructed via a non-trivial process that involves the values of the gates of the circuit on various inputs.

The rest of the paper is organized as follows. In Section 2 we give necessary definitions and collect technical statements. In Section 3 we study circuits computing the function with high correlation. In Section 4 we give bounds for randomized circuits. In Section 5 we study deterministic circuits. Finally, in Section 6 we give concluding remarks and state several open problems.

## 2 Definitions and Preliminaries

In this section we will give necessary definitions and collect technical statements that we will use throughout the paper.

We are going to study circuits computing the well known boolean majority function defined as follows:  $\text{MAJ}_n(x_1, x_2, \dots, x_n) = [\sum_{i=1}^n x_i \geq n/2]$ . Here,  $[\cdot]$  denotes the standard Iverson bracket: for a predicate  $P$ ,  $[P] = 1$  if  $P$  is true, and  $[P] = 0$  if  $P$  is false. To abuse notation, we will also use  $[m]$  to denote the set  $\{1, 2, \dots, m\}$ .

It will be convenient to use  $X = \{x_1, x_2, \dots, x_n\}$  for the set of  $n$  input bits. For an assignment  $A: X \rightarrow \{0, 1\}$ , by  $w(A)$  we denote the weight of  $A$ , that is,  $\sum_{x \in X} A(x)$ . For a subset of input variables  $S \subseteq X$ , by  $w_S(A)$  we denote the weight of  $A$  on  $S$ :  $w_S(A) = \sum_{x \in S} A(x)$ . By  $\text{MAJ}_S(X)$  we denote the majority function on  $S$ :  $\text{MAJ}_S(X) = [\sum_{x \in S} x \geq |S|/2]$ . In particular,  $\text{MAJ}_X$  is just  $\text{MAJ}_n$ .

An assignment  $A: X \rightarrow \{0, 1\}$  is called a minterm of  $\text{MAJ}_n$  if  $\text{MAJ}_n(A) = 1$ , but flipping any 1 to 0 in  $A$  results in an assignment  $A'$  such that  $\text{MAJ}_n(A') = 0$ . A maxterm is defined similarly with the roles of 0 and 1 interchanged.

The majority function is a special case of a threshold function:  $f(X) = [\sum_{i=1}^n a_i x_i \geq t]$ . For such a function  $f$  and an assignment  $A: X \rightarrow \{0, 1\}$ , let difference of  $f$  w.r.t.  $A$  be  $\text{diff}(f, A) = \sum_{i=1}^n a_i A(x_i) - t$ . In particular,  $f(A) = 1$  iff  $\text{diff}(f, A) \geq 0$ .

The  $\text{MAJ}_k \circ \text{MAJ}_k$  computational model that we study in this paper is defined as a depth two formula (we will call it a circuit also) consisting of arbitrary *threshold* gates of the form  $[\sum c_i x_i \geq t]$  where  $c_i$ 's are positive integers (this, in particular, means that the model is monotone) and  $\sum c_i \leq k$ . At the same time, abusing notation, by  $\text{MAJ}_n$  and  $\text{MAJ}_X$  we always mean the standard majority function. We note that the coefficients in  $c_i$  can be simulated by repetition of variables (note that  $k$  upper bounds the sum of the coefficients). So the generalization of the  $\text{MAJ}_k$  in the circuit compared to  $\text{MAJ}_n$  is that we allow arbitrary threshold. We note however, that if we are interested in the value of  $k$  up to a constant factor (which we usually do), it is not an actual generalization since any threshold can be simulated by substituting constants 0 and 1 as inputs to the circuit.

For a gate  $G$  at the bottom level of a  $\text{MAJ}_k \circ \text{MAJ}_k$  circuit, by  $X(G)$  we denote the set of its input bits.

## 2.1 Tail Bounds and Binomial Coefficients Estimates

We will use the following versions of Chernoff–Hoeffding bound (see, e.g., [3]).

► **Lemma 1** (Chernoff–Hoeffding bound). *Let  $Y = \sum_{i=1}^m Y_i$ , where  $Y_i$ ,  $i \in [m]$ , are independently distributed in  $[0, 1]$ . Then for all  $t > 0$ ,  $\Pr[Y > E[Y] + t], \Pr[Y < E[Y] - t] \leq e^{-2t^2/m}$ . For all  $\varepsilon > 0$ ,  $\Pr[Y > (1 + \varepsilon)E[Y]], \Pr[Y < (1 - \varepsilon)E[Y]] \leq e^{-\frac{\varepsilon^2}{3}E[Y]}$ .*

We will also need the following well known estimates for the binomial coefficients (see, e.g., [15, Section 4.2]):

► **Lemma 2.** *The middle binomial coefficient is about  $n^{1/2}$  times smaller than  $2^n$ . To make it smaller than  $2^n$  by arbitrary polynomial factor, it is enough to step away from the middle by about  $\Theta(\sqrt{n \ln n})$  ( $0 < c < 1$  is a constant below):*

$$\binom{n}{n/2} = \Theta(1) \cdot 2^n \cdot n^{-1/2} \quad \text{and} \quad \binom{n}{\frac{n}{2} + \frac{c\sqrt{n \ln n}}{2}} = \Theta(2^n n^{-\frac{1}{2}} n^{-\frac{c^2}{2}}). \quad (1)$$

## 2.2 Hypergeometric Distribution

The hypergeometric distribution is defined in the following way. Consider a set  $S$  of size  $m$  and its subset  $S'$  of size  $k$ . Select (uniformly) a random subset  $T$  of size  $t$  in  $S$ . Then a random variable  $|T \cap S'|$  has a hypergeometric distribution. The values  $m$ ,  $k$  and  $t$  are parameters here. We will need the following basic properties of this distribution.

► **Lemma 3.** *Suppose in hypergeometric distribution  $k = k(m) \leq m/2$  (that is,  $k$  may depend on  $m$ ). Let  $t = t(m)$  be a function with  $\varepsilon m < t < (1 - \varepsilon)m$  for some constant  $0 < \varepsilon < 1$ . Then, for any integer  $l$ ,  $\text{Prob}(|T \cap S'| = l) = O(k^{-1/2})$ , where  $O(\cdot)$  is for  $m \rightarrow \infty$  and the constant inside  $O(\cdot)$  depends on  $\varepsilon$ , but does not depend on  $m$ ,  $k$  and  $t$ . Moreover, if  $|l - \frac{tk}{m}| = O(1)$ , then this probability is in fact  $\Theta(k^{-1/2})$ .*

► **Lemma 4.** *Suppose in hypergeometric distribution  $k = k(m) \leq m/2$  (that is,  $k$  may depend on  $m$ ). Let  $t = t(m)$  be a function with  $\varepsilon m < t < (1 - \varepsilon)m$  for some constant  $0 < \varepsilon < 1$ . Consider an arbitrary antichain  $A$  on  $S'$  (that is, a family of subsets of  $S'$  none of which is a subset of some other). Then the probability  $\Pr[T \cap S' \subseteq A] = O(k^{-1/2})$ , where  $O(\cdot)$  is for  $m \rightarrow \infty$  and the constant inside  $O(\cdot)$  depends on  $\varepsilon$ , but does not depend on  $m$ ,  $k$  and  $t$ .*

► **Lemma 5.** *For  $S$ ,  $S'$  and  $T$  as above we have  $\text{Prob}\{|T \cap S'| \geq l\} \leq (tk/m)^l$ .*

## 3 Circuits with High Correlation

In this section, we prove that the minimum value of  $k$  for which there exists a  $\text{MAJ}_k \circ \text{MAJ}_k$  circuit that computes  $\text{MAJ}_n$  correctly on, say,  $2/3$  fraction of all the inputs, is equal to  $\Theta(n^{1/2})$ .

### 3.1 Upper Bound

► **Theorem 6.** *For any  $\varepsilon > 0$ , there exists a circuit  $C$  in  $\text{MAJ}_k \circ \text{MAJ}_k$ , where  $k = O_\varepsilon(n^{1/2})$ , that agrees with  $\text{MAJ}_n$  on at least  $(1 - \varepsilon)$  fraction of the boolean hypercube  $\{0, 1\}^n$ .*

**Proof Sketch.** The required circuit is straightforward: we just pick  $k$  random subsets  $S_1, S_2, \dots, S_k$  of  $X$  of size  $k$ , compute the majority for each of them, and then compute the majority of the results:  $C(X) = \text{MAJ}_k(\text{MAJ}_{S_1}(X), \text{MAJ}_{S_2}(X), \dots, \text{MAJ}_{S_k}(X))$ . The

resulting circuit has a high probability of error on middle layers of the boolean hypercube. We however select the parameters so that all the inputs from these middle layers constitute only a small  $\varepsilon/2$  fraction. We then show that among all the remaining inputs (not belonging to middle layers) there is only a fraction  $\varepsilon/2$  (of all the inputs) where  $\text{MAJ}_n$  may be computed incorrectly. Overall, this gives a circuit that errs on at most  $\varepsilon$  fraction of the inputs. ◀

### 3.2 Lower Bound

► **Theorem 7.** *Let  $C$  be a  $\text{MAJ}_k \circ \text{MAJ}_k$  circuit that computes  $\text{MAJ}_n$  correctly on a fraction  $1 - \varepsilon$  of all  $2^n$  inputs for a constant  $\varepsilon \leq 1/3$ . Then  $k = \Omega_\varepsilon(n^{1/2})$ .*

**Proof Sketch.** Let  $k = \alpha n^{1/2}$  for a small enough constant  $\alpha = \alpha(\varepsilon)$ . Note that such a circuit can read at most  $k^2 = \alpha^2 n$  of the input bits. This means that the circuit errs on a large number of inputs. ◀

## 4 Randomized Circuits

The upper bound from the previous section, however, is not enough to obtain a randomized circuit since the construction in Theorem 6 has a very high error probability on the middle layers of the boolean cube. By a randomized circuit here we mean a probabilistic distribution on deterministic circuits computing the function correctly on every input with high probability.

It is not difficult to see that the existence of a randomized circuit is equivalent to an existence of a deterministic circuit computing the function correctly on most of minterms and maxterms.

► **Lemma 8.** *If there exists a randomized circuit  $C$  in  $\text{MAJ}_k \circ \text{MAJ}_k$  computing  $\text{MAJ}_n$  with error probability  $\varepsilon$ , then there exists a deterministic circuit  $C$  in  $\text{MAJ}_k \circ \text{MAJ}_k$  computing  $\text{MAJ}_n$  incorrectly on at most  $\varepsilon$  fraction of minterms and maxterms. Conversely, if there exists a deterministic circuit  $C$  in  $\text{MAJ}_k \circ \text{MAJ}_k$  computing  $\text{MAJ}_n$  incorrectly on at most  $\varepsilon$  fraction of minterms and maxterms, then there exists a randomized circuit  $C$  in  $\text{MAJ}_k \circ \text{MAJ}_k$  computing  $\text{MAJ}_n$  with error probability at most  $2\varepsilon$ .*

From now on instead of probabilistic circuits we study deterministic circuits with high accuracy on two middle layers of  $\{0, 1\}^n$ .

### 4.1 Upper Bound

► **Theorem 9.** *There exists a randomized  $\text{MAJ}_k \circ \text{MAJ}_k$  circuit computing  $\text{MAJ}_n$  incorrectly on each input with probability at most  $1/\text{poly}(n)$  for  $k = O(n^{2/3} \log^{1/2} n)$ .*

**Proof Sketch.** Partition the set of  $n$  input bits into  $n^{1/3}$  blocks of size  $p = n^{2/3}$ :  $X = X_1 \sqcup X_2 \sqcup \dots \sqcup X_{\frac{n}{p}}$ . For each block  $X_i$ , compute  $[\sum_{x \in X_i} x \geq m]$  for all  $m \in [\frac{p}{2} - \frac{t}{2}, \frac{p}{2} + \frac{t}{2}]$  for  $t \approx n^{1/3} \log^{1/2} n$ , and return the majority of results. By selecting the right value of  $t$ , this gives a circuit that computes  $\text{MAJ}_n$  incorrectly only on a fraction  $\frac{1}{\text{poly}(n)}$  of inputs. ◀

### 4.2 Lower Bound

In this subsection we show that the upper bound of the previous subsection is essentially tight.

► **Theorem 10.** *If a  $\text{MAJ}_k \circ \text{MAJ}_k$  circuit computes  $\text{MAJ}_n$  on a  $1 - \varepsilon$  fraction of minterms and maxterms for  $\varepsilon < 1/10$ , then  $k = \Omega(n^{2/3})$ .*

**Proof Sketch.** The majority function have many inputs  $A \in \{0, 1\}^n$  with a property that changing a single bit in  $A$  changes the value of the function (these are precisely minterms and maxterms of  $\text{MAJ}_n$ ). If  $k = \alpha n^{2/3}$  for a small enough constant  $\alpha$ , a  $\text{MAJ}_k \circ \text{MAJ}_k$  circuit can reflect such a change in the value only for a small fraction of inputs. ◀

## 5 Deterministic Circuits

In this section, we consider  $\text{MAJ}_k \circ \text{MAJ}_k$  circuits that compute  $\text{MAJ}_n$  correctly on all  $2^n$  inputs.

### 5.1 Upper Bounds

#### 5.1.1 Depth Two

In this section, we present  $\text{MAJ}_k \circ \text{MAJ}_k$  circuits computing  $\text{MAJ}_n$  on all inputs for  $k = n - 2$  when  $n = 7, 9, 11$ . These circuits were found by extensive computer experiments (with the help of SAT-solvers). Though the examples below look quite “structured”, currently, we do not know how to generalize them to all values of  $n$  (not to say about constructing such circuits for sublinear values of  $k$ ). In the examples below, we provide  $k = n - 2$  sequences consisting of  $k = n - 2$  integers from  $[n]$ . These are exactly the input bits of the  $k$  majority gates at the lower level of the circuit. That is, each gate computes the standard  $\text{MAJ}_k$  function (whose threshold value is  $k/2$ ).

$n = 7:$	$n = 9:$	$n = 11:$
1 2 3 4 5	1 2 3 4 5 6 7	1 2 3 4 5 6 7 8 9
1 2 3 6 7	1 2 3 4 5 8 9	1 2 3 4 5 6 7 10 11
1 4 5 6 7	1 2 3 6 7 8 9	1 2 3 4 5 8 9 10 11
2 2 4 5 6	1 4 5 6 7 8 9	1 2 3 6 7 8 9 10 11
3 4 5 7 7	1 3 5 5 7 9 9	1 4 5 6 7 8 9 10 11
	1 2 4 6 6 8 8	1 2 2 4 6 6 8 10 10
	2 3 4 5 6 7 8	2 4 4 5 6 7 8 10 11
		3 3 5 5 7 7 8 9 11
		3 3 6 8 9 9 9 10 10

Note that in the examples above there is always a gate in the circuit having one variable repeated more than once. Next we observe that this is unavoidable for  $k = n - 2$ .

► **Lemma 11.** *For odd  $n$  there is no  $\text{MAJ}_k \circ \text{MAJ}_k$  circuit for  $k = n - 2$  with all gates being standard majorities (that is, with the threshold  $n/2$ ) and having exactly  $k$  distinct variables in each gate on the bottom level.*

#### 5.1.2 Depth Three

In this section we extend the proof of the upper bound for randomized depth-2 circuits (Theorem 9) to construct a circuit of depth 3 for  $k = O(n^{2/3})$  computing majority on all inputs.

► **Theorem 12.** *For  $k = O(n^{2/3})$  there is a circuit of depth 3 computing majority of  $n$  variables on all inputs.*

**Proof Sketch.** We adopt the strategy of the proof of Theorem 9. That is, we break inputs into  $O(n^{1/3})$  blocks, compute majorities on each block on middle  $O(n^{1/3})$  layers and then

compute the majority of the results. We use the third layer of majority gates to induce additional structure on the inputs. ◀

## 5.2 Lower Bound

In this section we will extend the lower bound on  $k$  above  $\Omega(n^{2/3})$  for depth-2 circuits computing  $\text{MAJ}_n$  on all inputs.

► **Theorem 13.** *Suppose a  $\text{MAJ}_k \circ \text{MAJ}_k$  circuit computes  $\text{MAJ}_n$  on all inputs. Then  $k = \Omega(n^{13/19} \cdot (\log n)^{-2/19})$ .*

We also show the following result for the special case of circuits with bounded weights.

► **Theorem 14.** *Suppose a  $\text{MAJ}_k \circ \text{MAJ}_k$  circuit computes  $\text{MAJ}_n$  on all inputs and uses only weights at most  $W$  in the gates. Then  $k = \Omega(n^{7/10} \cdot (\log n)^{-1/5} \cdot W^{-3/10})$ .*

In particular, we get the following corollary for circuits with unweighted gates.

► **Corollary 15.** *Suppose a  $\text{MAJ}_k \circ \text{MAJ}_k$  circuit computes  $\text{MAJ}_n$  on all inputs and each variable occurs in each gate of the bottom level at most once. Then  $k = \Omega(n^{7/10} \cdot (\log n)^{-1/5})$ .*

The rest of this section is devoted to the unified proof of these lower bounds. To follow this proof it is convenient to think that  $k = n^{\frac{2}{3} + \varepsilon}$  for some small  $\varepsilon > 0$ . In the end it will indeed be the case up to a logarithmic factor. In the proof we will calculate everything precisely in terms of parameters  $n$  and  $k$ , but we will provide estimates assuming that  $k = n^{2/3 + \varepsilon}$ . This is done in order to help the reader to follow the proof.

Let  $F$  be a  $\text{MAJ}_k \circ \text{MAJ}_k$  formula computing  $\text{MAJ}_n$  on all inputs from  $\{0, 1\}^n$ . Denote by  $W$  the largest weight of a variable in gates of  $F$ .

### 5.2.1 Normalizing a formula

We start by “normalizing”  $F$ , that is, removing some pathological gates from  $F$ . We do this in two consecutive stages.

**Stage 1: removing AND-like gates.** We will need that no gate can be fixed to 0 by assigning a small number of variables to 0 (here and in what follows we consider gates from the bottom level only). For this, assume that there is a gate that can be fixed to 0 by assigning to 0 less than  $n/(100k) = n^{1/3 - \varepsilon}/100$  variables. Take these variables and substitute them by 0; this kills this gate (and might potentially introduce new gates with the property). We repeat this process until there are no bad gates left. Recall that the number of gates at the bottom level is at most  $k = n^{2/3 + \varepsilon}$ , so there are at most  $k = n^{2/3 + \varepsilon}$  steps in this process and hence  $n$  is replaced by  $99n/100$ . To simplify the presentation, we just assume that  $|X| = n$  and that  $F$  has no bad gates.

**Stage 2: removing other pathological gates and variables.** The formula  $F$  contains at most  $k^2 = n^{\frac{4}{3} + 2\varepsilon}$  occurrences of variables (counting with multiplicities). Let  $x^* \in X$  be a least frequent variable at the leaves. The number of occurrences of  $x^*$  is at most  $k^2/n = n^{1/3 + 2\varepsilon}$ . In the following we consider only assignments  $A$  with  $\text{diff}(\text{MAJ}_n, A) = -1$  setting  $x^*$  to 0:

$$\mathcal{A}^* = \{A: X \rightarrow \{0, 1\} \mid \text{diff}(\text{MAJ}_n, A) = -1 \text{ and } A(x^*) = 0\}.$$

We also focus on the gates from the first level that depend on  $x^*$ , denote this set by  $\mathcal{G}^*$  (hence  $|\mathcal{G}^*| \leq k^2/n = n^{1/3 + 2\varepsilon}$ ). The total number of variables in the gates from  $\mathcal{G}^*$  (counting with multiplicities) is at most  $k|\mathcal{G}^*| \leq k^3/n = n^{1 + 3\varepsilon}$ .



We now additionally normalize the circuit. We get rid of the following bad gates and variables:

1. gates in  $\mathcal{G}^*$  that can be assigned to 1 by fixing less than  $n^2/(100k^2) = n^{2/3-2\varepsilon}/100$  variables in  $X \setminus \{x^*\}$  to 1;
2. gates in  $\mathcal{G}^*$  with the weight of the variable  $x^*$  greater than  $100k^3/n^2 = 100n^{3\varepsilon}$ ;
3. variables with total weight in all gates in  $\mathcal{G}^*$  greater than  $100k^3/n^2 = 100n^{3\varepsilon}$ .

We do this by the following iterative procedure. If on some step we have a gate violating 1 we fix less than  $n^2/(100k^2) = n^{2/3-2\varepsilon}/100$  variables of the gate among  $X \setminus \{x^*\}$  to 1 to assign the gate to a constant. If we have a gate violating 2 we fix all the variables of the gate among  $X \setminus \{x^*\}$  to 1 to assign the gate to a constant. If we have a variable violating 3, we fix the violating variable to 1.

We note that if we fix all variables in  $G \in \mathcal{G}^*$  except  $x^*$  to 1, then the gate becomes constant. Indeed, if it is not constant, then the gate outputs 0 on the input with  $x^* = 0$  and the rest of the variables equal to 1. Due to the monotonicity of the gate this means that the gate can be assigned to 0 by assigning a single variable  $x^*$  to 0 and we got rid of the gates with this property on the first stage of the normalization.

Since there are at most  $k^2/n = n^{1/3+2\varepsilon}$  gates in  $\mathcal{G}^*$  we will fix at most  $n/100$  variables for case 1. Since the total weight of  $x^*$  is at most  $k^2/n = n^{1/3+2\varepsilon}$  we will have case 2 at most  $n/(100k) = n^{1/3-\varepsilon}/100$  times. Since each gate has at most  $k = n^{2/3+\varepsilon}$  variables we will fix at most  $n/100$  variables for the second case. Since the total weight of all variables in  $\mathcal{G}^*$  is at most  $k^3/n = n^{1+3\varepsilon}$  we will fix at most  $n/100$  of them for the case 3.

In particular, we have fixed all variables having weight greater than  $100k^3/n^2 = 100n^{3\varepsilon}$  in some gate of  $\mathcal{G}^*$ , so from now on we can assume that  $W \leq 100k^3/n^2$ .

Another important observation is that now in each gate there are at least  $n^2/(100k^2)$  inputs. Otherwise the gate falls under condition of case 1 above.

After this normalization  $n$  is replaced by  $97n/100$ . To simplify the presentation, again, we assume that  $|X| = n$  and the circuit  $F$  is normalized. Note that after redefining  $n$  the threshold of the function  $\text{MAJ}_n$  we are computing is no longer  $n/2$ , but rather is  $cn$  for some constant  $c$  close to  $1/2$ . This does not affect the computations in the further proof.

## 5.2.2 Analysis

The key idea is that if we have an assignment  $A \in \mathcal{A}^*$  with  $\text{diff}(\text{MAJ}_n, A) = -1$ , then there is a gate  $G \in \mathcal{G}^*$  with  $-W \leq \text{diff}(G, A) \leq -1$ . Indeed, otherwise we can flip the variable  $x^*$ , the value of  $\text{MAJ}_n$  changes, but none of the gates changes their value. The plan of the proof is to construct an assignment that violates this condition. This will lead to a contradiction.

For an assignment  $A \in \mathcal{A}^*$  with  $\text{diff}(\text{MAJ}_n, A) = -1$  and integer parameters  $s$  and  $d$  (to be chosen later), consider the following process  $\text{walk}(A, s, d)$ .

- 1:  $A_0 \leftarrow A$
- 2: **for**  $i = 1$  to  $s$  **do**
- 3:   **if** for each  $G \in \mathcal{G}^*$ ,  $\text{diff}(G, A_{i-1}) \notin \{-d, -d+1, \dots, -1\}$  **then**
- 4:     stop the process
- 5:   **else**
- 6:      $G_i \leftarrow$  any gate from  $\mathcal{G}^*$  such that  $-d \leq \text{diff}(G, A_{i-1}) < 0$
- 7:      $X_i \leftarrow$  set of variables  $G_i$  depends on that are assigned 1 by  $A_{i-1}$
- 8:      $y_i \leftarrow$  a uniform random variable from  $X_i$
- 9:      $A_i \leftarrow$  assignment to  $X$  resulting from flipping the value of  $y_i$  in  $A_{i-1}$
- 10:   **end if**
- 11: **end for**

Clearly, this process decreases the weight of the initial assignment  $A$  by 1 at each iteration, for at most  $s$  iterations. In particular,  $w(A) - w(A_i) = i$ . We now consider three cases.

**Case 1.** *There exists an assignment  $A \in \mathcal{A}^*$  with  $\text{diff}(\text{MAJ}_n, A) = -1$  such that  $\text{walk}(A, s, d)$  stops after less than  $s$  iterations for some choices of random bits. This means that after  $t < s$  iterations, for all the gates  $G$  in  $\mathcal{G}^*$  we have that either  $\text{diff}(G, A_t) < -d$ , or  $\text{diff}(G, A_t) \geq 0$ .*

We select randomly a subset  $T$  of  $t$  variables from  $Z = \{x \in X \setminus \{x^*\}: A_t(x) = 0\}$  and flip them. Denote the resulting assignment by  $A'$ . Clearly,  $w(A) = w(A')$  and so  $\text{diff}(\text{MAJ}_n, A') = -1$ . Therefore there must be a gate  $G$  in  $\mathcal{G}^*$  such that  $-W \leq \text{diff}(G, A') < 0$ . Thus, before flipping  $t$  random variables, all the gates with negative difference has difference less than  $-d$ , while after the flipping, at least one gate  $G$  has difference at least  $-W$ . Let  $Z' = \{x \in X(G) \setminus \{x^*\}: A_t(x) = 0\}$ . This means that the flipping changed the values of at least  $r = (d - W)/W$  variables of  $G$ , that is,  $|T \cap Z'| \geq r$ .

Let  $p$  be the probability that  $|T \cap Z'| \geq r$  where the probability is taken over the random choice of  $T$ . By choosing the parameters  $s$  and  $d$  we will make  $p$  small enough so that with non-zero probability no gate from  $\mathcal{G}^*$  satisfies this. Due to the discussion above this leads to a contradiction since flipping  $x^*$  changes the value of the function, but not the value of the circuit. The probability that no gate from  $\mathcal{G}^*$  satisfies  $|T \cap Z'| \geq r$  is at least  $1 - |\mathcal{G}^*|p$ . The probability  $p$  can be upper bounded using Lemma 5:  $p \leq \left(\frac{t|Z'|}{|Z|}\right)^r \leq \left(\frac{sk}{n/2}\right)^r$  where the second inequality follows since  $t < s$ ,  $|Z'| \leq k$  and  $|Z| \geq \frac{n}{2}$ .

We want the probability  $1 - |\mathcal{G}^*|p$  to be positive. Since  $|\mathcal{G}^*| \leq k^2/n = n^{1/3+2\varepsilon}$  we get the following inequality on  $s$ ,  $d$ , and  $k$ :  $(k^2/n) \cdot (2sk/n)^r < 1$ . We can satisfy this if  $sk < n/4$  and  $r \geq \log \frac{k^2}{n}$ . Since  $\log n > \log \frac{k^2}{n}$  for the latter it is enough to have  $d = W \log n$ . Overall, this case poses the following constraint for the considered parameters:

$$sk \leq n/4. \tag{2}$$

**Case 2.** *For each assignment  $A \in \mathcal{A}^*$  (i.e.,  $\text{diff}(\text{MAJ}_n, A) = -1$ ) the process  $\text{walk}(A, s, d)$  goes through all  $s$  iterations for all choices of random bits. We consider two subcases here.*

**Case 2.1.** *For each assignment  $A \in \mathcal{A}^*$  (i.e.,  $\text{diff}(\text{MAJ}_n, A) = -1$ ) there exists a choice of variables  $y_1, \dots, y_s$  at line 8 of the process  $\text{walk}(A, s, d)$ , such that for each gate  $G \in \{G_1, \dots, G_s\}$  (recall that the gates  $G_1, \dots, G_s$  are selected at line 6 of the process) we have  $\text{diff}(G, A) \leq f$ , where  $f$  is again a positive parameter to be chosen later.*

We estimate the expected number  $E$  of gates  $G$  from  $\mathcal{G}^*$  that have  $-d \leq \text{diff}(G, A) \leq f$  where the expectation is taken over the random choices of  $A$ . Note that a particular gate  $G \in \mathcal{G}^*$  may appear in the sequence  $G_1, \dots, G_s$  at most  $d$  times: the first time it appears, it must have  $\text{diff}(G, A_1) \leq -1$  for the current assignment  $A_1$ , the next time it has  $\text{diff}(G, A_2) \leq -2$  for the new current assignment  $A_2$ , and so on. If  $Ed < s$  we get a contradiction: take an assignment  $A \in \mathcal{A}^*$  with  $\text{diff}(\text{MAJ}_n, A) = -1$  such that the number of gates  $G$  in  $\mathcal{G}^*$  with  $-d \leq \text{diff}(G, A) \leq f$  is at most  $E$ , then we cannot have that for all of  $G_1, \dots, G_s$  it is true that  $-d \leq \text{diff}(G_i, A) \leq f$ , there are just not enough gates with this diff.

Now we upper bound  $E$ . Due to the normalization stage any fixed gate has at least  $n^2/(100k^2) = n^{2/3-2\varepsilon}/100$  variables in it. Note that the set of inputs  $B$  to the gate  $G$  that give  $\text{diff}(G, B) = i$  for any  $i$  form an antichain. Then due to Lemma 4 the probability for a gate to attain a certain value is at most  $O(k/n) = O(1/n^{1/3-\varepsilon})$ .

Hence

$$E \leq |\mathcal{G}^*| \cdot (f + d) \cdot O\left(\frac{k}{n}\right) = \frac{k^2}{n} \cdot (f + d) \cdot O\left(\frac{k}{n}\right) = O\left(\frac{k^3(f + d)}{n^2}\right) = O\left(\frac{k^3 f}{n^2}\right),$$

where for the last equality we add the constraint

$$d = O(f). \tag{3}$$

Overall, this case poses the following constraint for the parameters:

$$O\left(\frac{k^3 f d}{n^2}\right) = O(f d n^{3\varepsilon}) < s. \tag{4}$$

**Case 2.2.** *There exists an assignment  $A \in \mathcal{A}^*$  (i.e.,  $\text{diff}(\text{MAJ}_n, A) = -1$ ) such that for any choice of variables  $y_1, \dots, y_s$ , for at least one gate  $G \in \{G_1, \dots, G_s\}$  we have  $\text{diff}(G, A) > f$ .*

Fix a gate  $G \in \mathcal{G}^*$  with  $\text{diff}(G, A) > f$ . We are going to upper bound the probability (over the random choices of variables  $y_1, \dots, y_s$ ) that  $G$  appears among  $G_1, \dots, G_s$  during the process. If this probability is less than  $1/k$ , then by the union bound with a positive probability no gate such gate appears among  $G_1, \dots, G_s$  which leads to a contradiction with the case statement.

For  $G$  to appear among  $G_1, \dots, G_s$ , the process has to select a variable appearing in  $G$  at line 8 many times. Indeed, if  $G$  appears in the process, then its  $\text{diff}$  with the current assignment is negative. At the same time, in the beginning of the process  $\text{diff}(G, A) > f$ . Each time when the process reduces a variable at line 8 (that is, changes its value from 1 to 0), the value of the linear function computed at  $G$  decreases by at most  $W$  (just because  $W$  is the maximum weight of a variable in all the gates in  $\mathcal{G}^*$ ). Thus, it is enough to upper bound the probability that for a fixed gate  $G \in \mathcal{G}^*$  with  $\text{diff}(G, A) > f$ , the process selects a variable from  $X(G)$  at least  $f/W$  times.

Let  $Y_1, \dots, Y_s$  be random 0/1-variables defined as follows:  $Y_i = 1$  iff the  $i$ -th reduced variable appears in  $G$  (i.e.,  $y_i \in X(G)$ ). Let  $Y = \sum_{i=1}^s Y_i$ . Our goal is to upper bound  $\text{Prob}(Y \geq f/W)$ .

Let  $H_1, \dots, H_l$  be all the gates that share at least one variable with  $G$ . Assume that on step  $j$  we reduce a variable from  $H_i$ . Then

$$\text{Prob}(Y_j = 1) = \text{Prob}(y_i \in X(G)) = \frac{|X(G) \cap X(H_i)|}{|\{x \in X(H_i) : A_{j-1}(x) = 1\}|}.$$

Due to the stage 2.1 of the normalization process,  $|\{x \in X(H_i) : A_{j-1}(x) = 1\}| \geq \frac{n^2}{100k^2} - d$ . To see this, assume the contrary. Recall that  $-d \leq \text{diff}(H_i, A_{j-1}) < 0$ . This means that by increasing at most  $d$  variables (i.e., changing their values from 0 to 1) from  $X(H_i)$  in  $A_{j-1}$  results in an assignment of weight at most  $\frac{n^2}{100k^2}$  that sets  $H_i$  to 1. This, in turn, contradicts to the fact that the circuit is normalized. Thus,

$$\text{Prob}(Y_j = 1) \leq \frac{|X(G) \cap X(H_i)|}{\frac{n^2}{100k^2} - d} \leq \frac{|X(G) \cap X(H_i)|}{\frac{n^2}{200k^2}},$$

where we add a constraint

$$d \leq \frac{n^2}{200k^2}. \tag{5}$$

We are now going to use the fact that variables from a fixed gate  $H_i$  can be reduced at most  $d$  times. We upper bound  $Y = \sum_{i=1}^s Y_i$  by the following random variable:  $Z = \sum_{i=1}^l \sum_{j=1}^d Z_{ij}$ , where each  $Z_{ij}$  is a random 0/1-variable such that

$$\text{Prob}(Z_{ij} = 1) = \frac{|X(G) \cap X(H_i)|}{\frac{n^2}{200k^2}},$$

and  $Z_{ij}$  are independent. That is, instead of reducing variables in some of  $H_i$ 's in some random order, we reduce  $d$  variables in each  $H_i$ . Thus we reduce maximal possible number of variables in all gates. Clearly, for any  $r$  we have  $\text{Prob}(Y \geq r) \leq \text{Prob}(Z \geq r)$ .

Let us bound the expectation of  $Z$ . Since due to the normalization each variable of  $G$  appear in other gates at most  $100k^3/n^2 = 100n^{3\varepsilon}$  times, we have

$$\sum_{i,j} |X(G) \cap X(H_i)| \leq d \cdot (100k^3/n^2) \cdot |X(G)| \leq 100 \cdot d \cdot k^4/n^2 = 100 \cdot n^{2/3+4\varepsilon} \cdot W \cdot \log n.$$

Overall we get  $EZ \leq \frac{100dk^4/n^2}{n^2/200k^2} = 4 \cdot 10^4 \cdot d \frac{k^6}{n^4} = 4 \cdot 10^4 \cdot n^{6\varepsilon} \cdot W \cdot \log n$ . Application of Chernoff–Hoeffding bound (Lemma 1) immediately implies that the probability that  $Z$  is twice greater than the expectation is exponentially small in  $d \cdot \frac{k^6}{n^4}$ . Since  $d \cdot \frac{k^6}{n^4} = W \cdot \log n \cdot n^{9\varepsilon}$  grows asymptotically faster than  $\log n$  for sure, we conclude that  $\text{Prob}(Z \geq 2 \cdot EZ) < \frac{1}{n} \leq \frac{1}{k}$ . Hence, if  $f/W \geq 2 \cdot EZ$ , then  $\text{Prob}(Y \geq f/W) \leq \text{Prob}(Z \geq 2 \cdot EZ) < \frac{1}{k}$  as desired. Overall, this gives us the following constraint:

$$f \geq 4 \cdot 10^4 \cdot d \cdot W \cdot \frac{k^6}{n^4} = 4 \cdot 10^4 \cdot n^{9\varepsilon} \cdot W^2 \cdot \log n. \quad (6)$$

### 5.2.3 Tuning the parameters

It remains to set the parameters so that the inequalities (2)–(6) are satisfied and  $k$  is as large as possible. The inequality (4) sets a lower bound on  $s$  in terms of  $f$ , while (6) sets a lower bound on  $f$ . Putting them together gives a lower bound on  $s$ :  $s \geq 4 \cdot 10^4 \cdot \frac{k^9}{n^6} \cdot W^3 \cdot \log^2 n$ . Combining it with the upper bound on  $s$  from (2), we can set the following equality on  $k$  and  $n$ :  $\frac{n}{4k} = 4 \cdot 10^4 \cdot \frac{k^9}{n^6} \cdot W^3 \cdot \log^2 n$ . Thus  $k = \Omega\left(\frac{n^{7/10}}{(\log n)^{1/5} W^{3/10}}\right)$  and it is easy to see that with this  $k$  we can pick other parameters to satisfy all the constraints (we set  $f$  so that (6) turns into an equality, the inequalities (3) and (5) are satisfied since  $W \leq \frac{k^3}{n^2}$ ).

This gives a proof of Theorem 14. For  $W = 1$  we get  $k = n^{7/10} \cdot (\log n)^{-1/5}$ , which gives a proof for Corollary 15. For unbounded  $W$  recall that we can assume  $W \leq \frac{k^3}{n^2}$  and thus  $k = n^{13/19} \cdot (\log n)^{-2/19}$  and Theorem 13 follows.

## 6 Conclusion and Open Problems

The most interesting question left open is whether one can prove non-trivial upper bounds for  $k$  in the worst case. Currently, we do not know how to construct  $\text{MAJ}_k \circ \text{MAJ}_k$  circuits computing  $\text{MAJ}_n$  on all inputs even for  $k = n - 2$  (though we have many examples of such circuits for  $n = 7, 9, 11$ ), not to say about  $k = n^\varepsilon$  for  $\varepsilon < 1$ .

Another natural open question is to get rid of the logarithmic gap between upper and lower bound for depth-2 randomized circuits.

A natural direction is to extend our studies to the case of non-monotone  $\text{MAJ}_k \circ \text{MAJ}_k$  circuits.

Many of our results naturally translate to larger depth circuits. Indeed, note that in the proofs of lower bounds we do not use the fact that the function on the top of the circuit is

majority. In these proofs it can be any monotone function. Thus we can split a depth- $d$  circuit consisting of  $\text{MAJ}_k$  into two parts: bottom layer and the rest of the circuit. Then our lower bounds translate to this setting straightforwardly. It is interesting to proceed with the studies of larger depth majority circuits.

**Acknowledgments.** We would like to thank the participants of Low-Depth Complexity Workshop (St. Petersburg, Russia, May 21–25, 2016) for many helpful discussions.

---

## References

- 1 Eric Allender and Michal Koucký. Amplifying lower bounds by means of self-reducibility. *J. ACM*, 57(3), 2010. doi:10.1145/1706591.1706594.
- 2 Xi Chen, Igor Carboni Oliveira, and Rocco A. Servedio. Addition is exponentially harder than counting for shallow monotone circuits. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:123, 2015. URL: <http://eccc.hpi-web.de/report/2015/123>.
- 3 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009. URL: <http://www.cambridge.org/gb/knowledge/isbn/item2327542/>.
- 4 Mikael Goldmann, Johan Hästad, and Alexander A. Razborov. Majority gates VS. general weighted threshold gates. *Computational Complexity*, 2:277–300, 1992. doi:10.1007/BF01200426.
- 5 Oded Goldreich. Valiant’s polynomial-size monotone formula for majority, 2001. Available at <http://www.wisdom.weizmann.ac.il/~oded/PDF/mono-maj.pdf>.
- 6 Thomas Hofmeister. The power of negative thinking in constructing threshold circuits for addition. In *Proceedings of the Seventh Annual Structure in Complexity Theory Conference, Boston, Massachusetts, USA, June 22-25, 1992*, pages 20–26, 1992. doi:10.1109/SCT.1992.215377.
- 7 Stasys Jukna. *Boolean Function Complexity – Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012. doi:10.1007/978-3-642-24508-4.
- 8 Stasys Jukna, Alexander A. Razborov, Petr Savický, and Ingo Wegener. On P versus NP cap co-NP for decision trees and read-once branching programs. *Computational Complexity*, 8(4):357–370, 1999. doi:10.1007/s000370050005.
- 9 Jesse Kamp and David Zuckerman. Deterministic extractors for bit-fixing sources and exposure-resilient cryptography. *SIAM J. Comput.*, 36(5):1231–1247, 2007. doi:10.1137/S0097539705446846.
- 10 Daniel M. Kane and Ryan Williams. Super-linear gate and super-quadratic wire lower bounds for depth-two and depth-three threshold circuits. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 633–643. ACM, 2016. URL: <http://dl.acm.org/citation.cfm?id=2897518>, doi:10.1145/2897518.2897636.
- 11 Frédéric Magniez, Ashwin Nayak, Miklos Santha, Jonah Sherman, Gábor Tardos, and David Xiao. Improved bounds for the randomized decision tree complexity of recursive majority. *Random Struct. Algorithms*, 48(3):612–638, 2016. doi:10.1002/rsa.20598.
- 12 Marvin Minsky and Seymour Papert. *Perceptrons – an introduction to computational geometry*. MIT Press, 1987.
- 13 Elchanan Mossel and Ryan O’Donnell. On the noise sensitivity of monotone functions. *Random Struct. Algorithms*, 23(3):333–350, 2003. doi:10.1002/rsa.10097.
- 14 Ryan O’Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014. URL: <http://www.cambridge.org/de/academic/subjects/computer-science/>

algorithmics-complexity-computer-algebra-and-computational-g/  
analysis-boolean-functions.

- 15 Robert Sedgewick and Philippe Flajolet. *An introduction to the analysis of algorithms*. Addison-Wesley-Longman, 1996.
- 16 Kai-Yeung Siu and Jehoshua Bruck. On the power of threshold circuits with small weights. *SIAM J. Discrete Math.*, 4(3):423–435, 1991. doi:10.1137/0404038.
- 17 Leslie G. Valiant. Short monotone formulae for the majority function. *J. Algorithms*, 5(3):363–366, 1984. doi:10.1016/0196-6774(84)90016-6.

# Minkowski Games\*

Stéphane Le Roux<sup>1</sup>, Arno Pauly<sup>2</sup>, and Jean-François Raskin<sup>3</sup>

- 1 Département d’Informatique, Université libre de Bruxelles, Brussels, Belgium  
stephane.le.roux@ulb.ac.be
- 2 Département d’Informatique, Université libre de Bruxelles, Brussels, Belgium  
arno.m.pauly@gmail.com
- 3 Département d’Informatique, Université libre de Bruxelles, Brussels, Belgium  
jraskin@ulb.ac.be

---

## Abstract

We introduce and study Minkowski games. In these games, two players take turns to choose positions in  $\mathbb{R}^d$  based on some rules. Variants include boundedness games, where one player wants to keep the positions bounded (while the other wants to escape to infinity), and safety games, where one player wants to stay within a given set (while the other wants to leave it).

We provide some general characterizations of which player can win such games, and explore the computational complexity of the associated decision problems. A natural representation of boundedness games yields coNP-completeness, whereas the safety games are undecidable.

**1998 ACM Subject Classification** F.3.1 Specifying and Verifying and Reasoning about Programs

**Keywords and phrases** Control in  $\mathbb{R}^d$ , determinacy, polytopic/arbitrary, coNP-complete, undecidable

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.50

## 1 Introduction

**Minkowski games.** In this paper we define and study *Minkowski games*. A *Minkowski play* is an infinite duration interaction between two players, called **Player A** and **Player B**, in the space  $\mathbb{R}^d$ . A *move* in a Minkowski play is a subset of  $\mathbb{R}^d$ . **Player A** has a set of moves  $\mathcal{A}$  and **Player B** has a set of moves  $\mathcal{B}$ . The play starts in a position  $a_0 \in \mathbb{R}^d$  and is played for an infinite number of rounds as follows. For a round starting in position  $a$ , **Player A** chooses  $A \in \mathcal{A}$  and **Player B** chooses a vector  $b$  in  $a + A$ , where  $+$  denotes the Minkowski sum. Next, **Player B** chooses  $B \in \mathcal{B}$  and **Player A** chooses a vector  $a'$  in  $b + B$ . Then a new round starts in the position  $a'$ . The *outcome* of a Minkowski play is thus an infinite sequence of vectors  $a_0 b_0 a_1 b_1 \dots a_n b_n \dots$  obtained during this infinite interaction. Each outcome is either winning for **Player A** or for **Player B**, and this is specified by a winning condition.

We consider two types of winning conditions. First, we consider *boundedness*: an outcome  $a_0 b_0 a_1 b_1 \dots a_n b_n \dots$  is winning for **Player A** in the boundedness game if there is a bounded subset  $\text{Safe} \subseteq \mathbb{R}^d$  such that the outcome stays in **Safe**, *i.e.* for all  $i \geq 0$ ,  $a_i \in \text{Safe}$  and  $b_i \in \text{Safe}$ , otherwise it is winning for **Player B**. Second, we consider *safety*: given a subset  $\text{Safe} \subseteq \mathbb{R}^d$ , an outcome is winning for **Player A** if it stays in **Safe**, otherwise **Player B** wins.

In this paper  $\mathcal{A}$  and  $\mathcal{B}$  are finite sets  $\{A_1, A_2, \dots, A_{n_A}\}$  and  $\{B_1, B_2, \dots, B_{n_B}\}$ , and both the moves and **Safe** in the safety Minkowski games are bounded. A long version [16] of this

---

\* This work was supported by the ERC inVEST (279499) project.



paper, providing comprehensive lemmas and proofs, relaxes these assumptions when possible and gives counterexamples otherwise.

The Minkowski games are a natural mathematical abstraction to model the interaction between two agents taking actions, modeled by moves, with imprecision as the adversary resolves nondeterminism by picking a vector in the move chosen by the other player.<sup>1</sup> Perhaps more importantly, the appeal of Minkowski games comes also from their simple and natural definition. We provide in this paper general results about these games and study several of their incarnations in which moves are given as (i) bounded polyhedral sets, (ii) sets defined using the first-order theory of the reals, or (iii) represented as compact or overt sets as defined in computable analysis [17]. Our results are as follows.

**Results.** We establish a necessary and sufficient condition for Player A to have a winning strategy in a *boundedness Minkowski game* with finitely many moves and give a simple proof (in comparison with Borel determinacy) that these games are determined (Theorem 5). We then turn our attention to computational complexity aspects of determining the winner of a game, *i.e.* who has a winning strategy. The necessary and sufficient condition that we have identified leads to a CONP solution when the moves are given as bounded rational polyhedral sets and we provide matching lower bounds (Theorem 6). These results hold both for moves represented by sets of linear inequalities and moves represented as the convex hulls of a finite set of points. Additionally, we show that for every fixed dimension  $d$ , determining the winner can be done in polynomial time (Corollary 11). When the moves are defined using the first-order theory of the reals, determining the winner of a boundedness game is shown to be 2EXPTIME-complete (Proposition 12). Finally, in the computable analysis setting the problem is semi-decidable (Proposition 14), and this is the best that we can hope for.

We characterize the set of winning positions in a *safety Minkowski game* as the greatest fixed point of an operator that removes the points where Player B can provably win (in finitely many rounds). We show that this greatest fixed point can be characterized by an approximation sequence of at most  $\omega$  steps (Proposition 17). This leads to semi-decidability in the general setting of computable analysis (Proposition 18). Then we show that identifying the winner in a safety Minkowski game is undecidable even for finite sets of moves that are given as bounded rational polyhedral sets (Theorem 19).

**Motivations and related works.** Infinite duration games are commonly used as mathematical framework for modeling the controller synthesis problem for reactive systems [15]. For reactive systems embedded in some physical environment, games played on hybrid automata have been considered, see e.g. [9] and references therein. In such a model, one controller interacts with an environment whose physical properties are modeled by valuations of  $d$  real-valued variables (vectors in  $\mathbb{R}^d$ ). Most of the problems related to the synthesis of controllers for hybrid automata are undecidable [9]. Restricted subclasses with decidable properties, such as timed automata and initialized rectangular automata have been considered [11, 8]. Most of the undecidability properties of those models rely on the coexistence of continuous and discrete evolutions of the configurations of hybrid automata. The one-sided version of the model of Minkowski games (where  $\mathcal{B} = \{\{0\}\}$ ) can be seen as a restricted form of an hybrid games in which each continuous evolution is of a unique fixed duration and space independent (such as in linear and rectangular hybrid automata). It is usually called discrete

---

<sup>1</sup> See further discussions on the practical appeal of these games for modeling systems evolving in multi-dimensional spaces when we report on related works.



time control in this setting. To the best of our knowledge, the closest models to Minkowski games that have been considered in the literature so far are Robot games defined by Doyen et al. in [7] and Bounded-Rate Multi-Mode Systems introduced by Alur et al. in [2, 1]. Minkowski games generalize robot games: there the moves are always singletons given as integer vectors. While we show that the Safety problem is undecidable for bounded safety objectives, it is easy to show that this problem is actually decidable for robot games. However, in [7] they investigate reachability of a specific position rather than safety conditions as we do here. Reachability was proven undecidable in [13] even for two-dimensional robot games. Boundedness objectives have not been studied for Robot games.

Bounded-Rate Multi-Mode Systems (BRMMS) are a restricted form of hybrid systems that can switch freely among a finite number of modes. The dynamics in each mode is specified by a bounded set of possible rates. The possible rates are either given by convex polytopes or as finite set of vectors. There are several differences with Minkowski games. First BRMMS are asymmetric and are thus closer to the special case of one-sided Minkowski games. Second, the actions in BRMMS are given by a mode and a time delay  $\delta \in \mathbb{R}$  while the time elapsing in our model can be seen as uniform and fixed. The ability to choose delays that are as small as needed makes the safety control problem for BRMMS with modes given as polytopes decidable while we show that the safety Minkowski games with moves defined by polytopes are undecidable. The discrete time control of BRMMS, which is more similar to the safety Minkowski games, has been solved only for modes given as finite sets of vectors and left open for modes given as polytopes. Our undecidability results can be easily adapted to the discrete time control of BRMMS and thus solves the open question left in that paper. Boundedness objectives have not been studied for BRMMS.

## 2 Preliminaries

**Linear constraints.** Let  $d \in \mathbb{N}_{>0}$ , and  $X = \{x_1, x_2, \dots, x_d\}$  be a set of variables. A *linear term* on  $X$  is a term of the form  $\alpha_1 x_1 + \dots + \alpha_d x_d$  where  $x_i \in X$ ,  $\alpha_i \in \mathbb{R}$  for all  $i$ ,  $1 \leq i \leq d$ . A *linear constraint* is a formula  $\alpha_1 x_1 + \dots + \alpha_d x_d \sim c$ , where  $\sim \in \{<, \leq, =, \geq, >\}$ , that compares a linear term with a constant  $c \in \mathbb{R}$ . Given a valuation  $v : X \rightarrow \mathbb{R}$ , amounting to a vector in  $\mathbb{R}^d$ , we write  $v \models \alpha_1 x_1 + \dots + \alpha_d x_d \sim c$  iff  $\alpha_1 v(x_1) + \dots + \alpha_d v(x_d) \sim c$ . Given a linear constraint  $\phi \equiv \alpha_1 x_1 + \dots + \alpha_d x_d \sim c$ , we write  $\llbracket \phi \rrbracket = \{v \in \mathbb{R}^d \mid v \models \alpha_1 x_1 + \dots + \alpha_d x_d \sim c\}$ . The constraint is called rational, if  $c$  and all the  $\alpha_i$  are in  $\mathbb{Q}$ .

**Polyhedra, polytopes, convex hull.** Given a finite set  $\mathcal{H} = \{\phi_1, \phi_2, \dots, \phi_n\}$  of linear constraints, we note  $\llbracket \mathcal{H} \rrbracket = \{v \in \mathbb{R}^d \mid \forall \phi \in \mathcal{H} : v \models \phi\}$  the set of vectors that satisfies all the linear constraints in  $\mathcal{H}$ . Such a set is a convex set and is usually called a *polyhedra*. In the special case that is bounded, then it is called a *polytope*. When a polytope is *closed*, then it is well-known that it can not only be represented by a finite set of linear inequalities that are all non-strict but also as the *convex hull* of a finite set of (extremal) vectors. The convex hull of a subset of a  $\mathbb{R}$ -vector space is noted and defined as follows:

$$\text{CH}(\mathcal{V}) := \left\{ \sum_{i=0}^n \alpha_i x_i \mid n \in \mathbb{N} \wedge \sum_{i=0}^n \alpha_i = 1 \wedge \forall i (x_i \in \mathcal{V} \wedge \alpha_i \geq 0) \right\}.$$

Carathéodory's theorem says that for all  $\mathcal{V} \subseteq \mathbb{R}^d$ , every point in  $\text{CH}(\mathcal{V})$  is a convex combination of at most  $d + 1$  points from  $\mathcal{V}$ . As a consequence, the  $n$  ranging over  $\mathbb{N}$  in the definition of the convex hull can safely be replaced with fixed  $d$ .

Let  $P$  be a closed polytope.  $P$  has two families of representations: its  $H$ -representations are the finite sets of linear inequalities  $\mathcal{H}$  such that  $\llbracket \mathcal{H} \rrbracket = P$ , and its  $V$ -representations are the finite sets of vectors  $\mathcal{V}$  such that  $\text{CH}(\mathcal{V}) = P$ . Some algorithmic operations are easier to perform on one representation or on the other. In general there is no polynomial time translation from one representation to the other unless  $P = NP$ . Nevertheless, such a polynomial time translation exists for fixed dimension. (We denote by  $\text{Ver}(P)$  the extremal points, *i.e.*, the vertices of a polytope  $P$ . It is the minimal set whose convex hull equals  $P$ .)

► **Theorem 1** ([4]). *Let  $d \in \mathbb{N}$ . There exists a polynomial time algorithm that given a  $H$ -representation of a rational closed polytope  $P$  of dimension  $d$ , computes a  $V$ -representation of  $P$ , and conversely, there exists a polynomial time algorithm that given a  $V$ -representation of  $P$ , computes a  $H$ -representation of  $P$ .*

**Minkowski sum.** For subsets  $A, B \in \mathbb{R}^d$  their Minkowski sum  $A + B$  is defined as  $\{a + b \mid a \in A \wedge b \in B\}$ . The Minkowski sum inherits commutativity and associativity from the usual sum of vectors. The set  $\{0\}$  is the neutral element, but there are no inverse elements in general. If  $A = \{a\}$  then  $A + B$  (resp.  $B + A$ ) is written  $a + B$  (resp.  $B + a$ ) in a slight abuse of notation. It is straightforward to prove that  $\text{CH}(A) + \text{CH}(B) = \text{CH}(A + B)$ . Especially, if  $A$  and  $B$  are convex, so is  $A + B$ . While  $A + A$  may be a strict superset of  $2A := \{2a \mid a \in A\}$  in general, for convex  $A$  we find  $A + A = 2A$ .

**Minkowski games – Strategies.** We have described in the introduction how the players interact in Minkowski games by choosing in each round a move and by resolving nondeterminism among the moves chosen by the other player. We now formally define the notions of strategies for each player, and the associated outcomes. When playing Minkowski games, players are applying *strategies*. In a game with moves  $\mathcal{A}$  and  $\mathcal{B}$ , strategies for the two players are defined as follows. A strategy for Player A is a function

$$\lambda_A : (\mathbb{R}^d)^* \rightarrow (\mathcal{A} \cup (\mathbb{R}^d)^*) \times \mathcal{B} \rightarrow \mathbb{R}^d$$

that respects the following consistency constraint: for all finite sequences of positions  $\rho \in (\mathbb{R}^d)^*$  that ends in  $v \in \mathbb{R}^d$ , and moves  $B \in \mathcal{B}$ ,  $\lambda_A(\rho, B) \in v + B$ . Symmetrically, a strategy for Player B is a function  $\lambda_B : (\mathbb{R}^d)^* \rightarrow (\mathcal{B} \cup (\mathbb{R}^d)^*) \times \mathcal{A} \rightarrow \mathbb{R}^d$  with the symmetric consistency constraint. The play  $a_0 b_0 a_1 b_1 \dots a_n b_n \dots$  induced by the strategies  $\lambda_A$  and  $\lambda_B$  is defined inductively by (for all  $i \geq 0$ )

$$b_i := \lambda_B(a_0 b_0 a_1 b_1 \dots a_i, \lambda_A(a_0 b_0 a_1 b_1 \dots a_i)) \text{ and } a_{i+1} := \lambda_A(a_0 b_0 \dots a_i b_i, \lambda_B(a_0 b_0 \dots a_i b_i)).$$

**Winning conditions and variants of Minkowski games.** By fixing the rule that determines who wins a Minkowski play, we obtain *Minkowski games*. Here we consider three types of Minkowski games, as below.

► **Definition 2.** A *boundedness Minkowski game* is a pair  $\langle \mathcal{A}, \mathcal{B} \rangle$  of sets of moves in  $\mathbb{R}^d$  for Player A and Player B. A play in a boundedness Minkowski game starts in some irrelevant  $a_0 \in \mathbb{R}^d$ , and the resulting play  $a_0 b_0 a_1 b_1 \dots a_n b_n \dots$  is winning for Player A if there exists a bounded subset **Safe** of  $\mathbb{R}^d$  such that  $a_i, b_i \in \text{Safe}$  for all  $i \in \mathbb{N}$ , otherwise Player B wins the game. The associated decision problem asks if Player A has a strategy  $\lambda_A$  which is winning against all the strategies  $\lambda_B$  of Player B.

► **Definition 3.** A *safety Minkowski game* is defined by  $\langle \mathcal{A}, \mathcal{B}, \text{Safe}, a_0 \rangle$ , where  $\mathcal{A}$  and  $\mathcal{B}$  are sets of moves for Player A and Player B,  $\text{Safe} \subseteq \mathbb{R}^d$  (bounded unless stated otherwise),

and  $a_0 \in \text{Safe}$  is the initial position. A play in a safety Minkowski game starts in  $a_0$ , and the resulting play  $a_0 b_0 a_1 b_1 \dots a_n b_n \dots$  is winning for Player A if  $a_i, b_i \in \text{Safe}$  for all  $i \in \mathbb{N}$ , otherwise Player B wins the game. The associated decision problem asks if Player A has a strategy  $\lambda_A$  which is winning against all the strategies  $\lambda_B$  of Player B.

A game is *single-sided* if  $\mathcal{B} = \{\{0\}\}$ , i.e. if Player B has only one trivial move. We use single-sided games to show that several of our lower-bounds hold for this subclass of games.

### 3 General results on the boundness problem

We first consider the special case of one-sided boundedness Minkowski games and provide a sufficient (and necessary) condition for Player A to win. The proof showcases some ideas that we use to fully characterize the general case. The general characterization in particular implies that the condition for the one-sided case is necessary.

► **Proposition 4.** We consider a one-sided boundedness Minkowski game  $\langle \mathcal{A}, \{0\} \rangle$  where  $\mathcal{A} = \{A_1, \dots, A_n\}$  and such that  $0 \in \text{CH}((x_i)_{1 \leq i \leq n})$  for all tuples  $(x_i)_{1 \leq i \leq n}$  in  $A_1 \times \dots \times A_n$ . Then Player A wins the boundedness game.

**Proof.** We describe the current state by some list of pairs  $(x_i, \alpha_i)_{i \leq n}$  such that  $x_i \in A_i$  and  $\alpha_i \in [0, 1]$ . We keep two invariants satisfied throughout the play: First, it will always be the case that the current position is equal to  $\sum_{i \leq n} \alpha_i x_i$ , which by boundedness of each  $A_i$  implies that Player A wins. Second, we maintain the invariant that there is some  $k \leq n$  with  $\alpha_k = 0$ . Initially, the choice of the  $x_i$  is arbitrary, and all  $\alpha_i$  are 0. This ensures that the strategy we describe for Player A is well-defined.

On his turn, Player A plays some  $A_k$  for  $k$  with  $\alpha_k = 0$ . Player B reacts with some  $x'_k \in A_k$ , and we set  $x_k := x'_k$  and  $\alpha_k := 1$ .

If immediately after the move, no  $\alpha_i$  is currently 0, we write a convex combination  $0 = \sum_{i \leq n} \beta_i x_i$ , which is possible by assumption. Let  $r := \max_{i \leq n} \frac{\beta_i}{\alpha_i}$ , and then update  $\alpha_i = \alpha_i - r^{-1} \beta_i$ . By the choice of the  $\beta_i$ , this leaves  $\sum_{i \leq n} \alpha_i x_i$  unchanged. The choice of  $r$  ensures that  $\alpha_i \in [0, 1]$  remains true, and more over, after the updating process, there is some  $k \leq n$  with  $\alpha_k = 0$ . Thus, the invariant is true again after the updating process. ◀

If  $0 \notin \text{CH}((x_i)_{i \leq m})$  for some  $x_i \in A_i$ , then Player B could win by playing  $x_i$  as response to a move  $A_i$  by Player A. If  $w$  is the shortest vector from 0 to  $\text{CH}((x_i)_{i \leq m})$ , then that strategy ensures that after any round, the position has moved by at least  $|w|$  in direction  $w$  – thus, the positions will leave any bounded region eventually.

We introduce some notation to state the general case. For a set of moves  $\mathcal{B}$  let  $\text{CH}(\mathcal{B}) := \{\text{CH}(B) \mid B \in \mathcal{B}\}$  and  $\overline{\mathcal{B}} := \{\overline{B} \mid B \in \mathcal{B}\}$ , where  $\overline{B}$  is the topological closure of  $B$ . We say that a strategy for Player B in a Minkowski game is *simple*, if it prescribes choosing always the same  $B \in \mathcal{B}$ , and if the choice  $a_i \in A_i$  depends only on the choice of  $A_i \in \mathcal{A}$  by Player A.

#### ► Theorem 5.

- *Boundedness Minkowski games are determined;*
- *the winner is the same for  $\langle \mathcal{A}, \mathcal{B} \rangle$  and  $\langle \text{CH}(\overline{\mathcal{A}}), \text{CH}(\overline{\mathcal{B}}) \rangle$ ;*
- *if Player B has a winning strategy, she has a simple one;*
- *Player A wins iff  $0 \in (\text{CH}\{a_i \mid i \leq |\mathcal{A}|\}) + \text{CH}(\overline{\mathcal{B}})$  for all  $(a_i)_{i \leq |\mathcal{A}|}$  with  $a_i \in A_i$  and  $B \in \mathcal{B}$ .*

#### 4 Computational complexity of the boundedness problem

In Section 3, we have provided general results on boundedness Minkowski games. Here we study the computational complexity of the associated decision problem <sup>2</sup>. To formulate complexity results, we need to consider classes of games that are defined in some effective way. Here we consider three ways to represent the sets of moves: by finite sets of linear constraints (or convex hulls of finite sets of vectors), by formulas in the first-order theory of the reals (that strictly extend the expressive power of linear constraints), and as compact sets or overt sets (closed sets with positive information) in the sense of computable analysis.

**Moves defined by linear constraints or as convex hulls.** Here we prove the following by reducing the 3-SAT problem to the complement of the boundedness Minkowski game problem.

► **Theorem 6.** *Given a boundedness Minkowski game  $\langle \mathcal{A}, \mathcal{B} \rangle$  where moves in the sets of moves  $\mathcal{A}$  and  $\mathcal{B}$  are defined by finite sets of rational linear constraints or as convex hulls of a finite sets of rational vectors, deciding the winner is CONP-COMplete. The hardness already holds for one-sided boundedness games.*

First note that Theorem 5 implies the useful Corollary 7, where a simple strategy  $\lambda_B$  for Player B is called a *vertex strategy*, if the  $a_i \in A_i$  chosen by  $\lambda_B$  are always vertices of  $A_i$ .

► **Corollary 7.** *If Player B has a winning strategy in a boundedness Minkowski game  $\langle \mathcal{A}, \mathcal{B} \rangle$  with closed moves in  $\mathcal{A}$  then she has a winning vertex strategy.*

**Proof.** By Theorem 5 he has a simple strategy, and even a vertex one since  $\text{CH}(\text{Ver}(P)) = P$  by definition. ◀

An important ingredient of the reduction below is a consequence of Corollary 7 and the determinacy of boundedness Minkowski games (Theorem 5): to show that Player A has a winning strategy, it suffices to show that he can spoil all the vertex strategies of Player B.

► **Lemma 8.** *There is a polynomial time reduction from the 3SAT problem to the complement of the boundedness problem for one-sided Minkowski games with closed, polytopic moves.*

**Proof.** Before going into the details of the proof, let us point out that the proof that we provide below works for both the  $H$ -representation and the  $V$ -representation. This is because the moves that we need to construct are all the convex hull of exactly three vectors. So the  $H$ -representation of such a convex hull can be obtained in polynomial time.

Let  $\Psi = \{C_1, C_2, \dots, C_n\}$  be a set of clauses with 3 literals defined on the set of Boolean variables  $X = \{x_1, x_2, \dots, x_m\}$ . Each  $C_i$  is of the form  $\ell_{i1} \vee \ell_{i2} \vee \ell_{i3}$  where each  $\ell_{ij}$  is either  $x$  or  $\neg x$  with  $x \in X$ .

To define the set of moves  $\mathcal{A}$  for Player A, we associate a move  $A_i$  with each clause  $C_i$ . The move is a subset of  $\mathbb{R}^d$ , where  $d = 2 \cdot |X| = 2 \cdot m$ , defined from  $C_i$  as follows. We associate with each variable  $x_k \in X$  two dimensions of  $\mathbb{R}^d$ :  $2k - 1$  and  $2k$ , and to each literals  $\ell_{ij}$  a vector noted  $\text{Vect}(\ell_{ij})$  defined as follows. If the literal  $\ell_{ij} = x_k$ , then the vector  $\text{Vect}(\ell_{ij})$  has zeros everywhere but in dimension  $2k - 1$  and  $2k$  where it is respectively equal to 1 and  $-1$ . If the literal  $\ell_{ij} = \neg x_k$ , then the vector  $\text{Vect}(\ell_{ij})$  has zeros everywhere but in dimension  $2k - 1$  and  $2k$  where it is respectively equal to  $-1$  and 1. So, it is the case that for all literals

<sup>2</sup> For all our complexity results, all the encoding of numbers and vectors that we use are the natural ones, i.e. integer or rational numbers are encoded succinctly in binary.

$\ell_1$  and  $\ell_2$ ,  $\text{Vect}(\ell_1) + \text{Vect}(\ell_2) = \mathbf{0}$  if and only if  $\ell_1 \equiv \neg\ell_2$ . Finally, the move associated with the clause  $C_i = \ell_{i1} \vee \ell_{i2} \vee \ell_{i3}$  is

$$A_i = \text{CH}(\text{Vect}(\ell_{i1}), \text{Vect}(\ell_{i2}), \text{Vect}(\ell_{i3})).$$

We now establish the correctness of our reduction. **First**, we show that if  $\Psi$  is satisfiable then Player B has a winning strategy in the boundedness game.

Let  $f : X \rightarrow \{0, 1\}$  be a valuation for the variables in  $X$  such that  $f \models \Psi$ . We associate with  $f$  a vertex strategy  $\lambda_B^f$  of Player B as follows. Because  $f \models \Psi$ , we know that in each clause  $C_i$ , there is a literal  $\ell_{ij}$  such that  $f \models \ell_{ij}$ . Then whenever Player A chooses moves  $A_i$ ,  $\lambda_B^f$  instructs Player B to choose  $\text{Vect}(\ell_{ij})$ . We now claim that this strategy is winning for Player B. The argument is as follows.

Because  $f \models \Psi$  and by definition of  $\lambda_B^f$ , it is the case that all the vertices chosen by  $\lambda_B^f$  are not associated with opposite literals. As a consequence, none of the vectors that will be played by Player B are opposite and so after  $k$  rounds in the game, at least one of the dimensions in the current position (assuming we started in position  $\mathbf{0}$ ) has absolute value larger than  $\frac{k}{m}$  and so there is no bounded set that can contain all the visited positions and Player B wins the boundedness game.

**Second**, we show that if  $\Psi$  is not satisfiable then Player B has no winning strategy. By Corollary 7, it is equivalent to show that Player B has no winning vertex strategies. We next prove that all the vertex strategies of Player B can be spoiled by Player A.

Remember that moves  $\mathcal{A} = \{A_1, A_2, \dots, A_n\}$  have been defined starting from the clauses  $C_1, C_2, \dots, C_n$ . Let  $\lambda_B^v$  be a vertex strategy of Player B, i.e. for all  $i$ ,  $1 \leq i \leq n$ ,  $\lambda_B^v(A_i) \in \text{Ver}(A_i)$ . We claim that for all such vertex strategy  $\lambda_B^v$ , there exists  $i_1$  and  $i_2$ ,  $1 \leq i_1 < i_2 \leq n$  such that  $\lambda_B^v(A_{i_1}) = -\lambda_B^v(A_{i_2})$ . If this is the case, then if Player A always alternates between move  $A_{i_1}$  and move  $A_{i_2}$ , then clearly the play remains in a bounded zone that contains  $\mathbf{0}$  as the value of all the dimensions is within the interval  $[-1, 1]$ .

For the sake of contradiction, let us assume that it is not the case that there exists  $i_1$  and  $i_2$ ,  $1 \leq i_1 < i_2 \leq n$  such that  $\lambda_B^v(A_{i_1}) = -\lambda_B^v(A_{i_2})$  and let us derive a contradiction. To obtain the contradiction, we note that the choice of one vertex per move corresponds to the choice of one literal per clause in the SAT problem. But if there is no  $i_1$  and  $i_2$ ,  $1 \leq i_1 < i_2 \leq n$  such that  $\lambda_B^v(A_{i_1}) = -\lambda_B^v(A_{i_2})$ , this means that we can choose one literal per clause so that we never choose two literals that are opposite. But this is only possible when  $\Psi$  is satisfiable, so we obtain our contradiction.  $\blacktriangleleft$

► **Lemma 9.** *Negative instances of the boundedness Minkowski games expressed with moves defined as sets of linear inequalities or convex hull of finite sets of vectors can be recognized by a nondeterministic polynomial time Turing machine.*

**Proof Sketch.** To show that Player A has no winning strategy, by Theorem 5, it suffices to exhibit  $a_1 \in \text{Ver}(A_1)$ ,  $a_2 \in \text{Ver}(A_2)$ ,  $\dots$ ,  $a_{n_A} \in \text{Ver}(A_{n_A})$ , and one  $B \in \mathcal{B}$ , such that

$$\mathbf{0} \notin \text{CH}(a_1, a_2, \dots, a_{n_A}) + \overline{B}.$$

Checking this can be done in polynomial time by reducing the testing to the satisfiability of a set of linear constraints.  $\blacktriangleleft$

**Fixed dimension and polytopical moves.** Here we show that given  $d \in \mathbb{N}$ , deciding the winner of a boundedness Minkowski game with polytopical moves in  $\mathbb{R}^d$  can be done in deterministic polynomial time. It relies on quick (for a fixed  $d$ ) translations from  $V$ -representations of

(closed) polytopes into  $H$ -representations, and vice-versa (see Theorem 1), so w.l.o.g. we focus here on the  $V$ -representation of polytopes. In this setting, by Theorem 5 it suffices to consider games with finite moves, since the extremal points of a polytope are finitely many. The degree of the polynomial (upper-)bounding the algorithmic complexity will be  $2d + 2$  in general, and  $d + 1$  for single-sided games. By Theorem 5 again, Player B has a winning strategy iff there exist  $a_1 \in A_1, \dots, a_n \in A_n$  (the moves in  $\mathcal{A}$ ) and  $B \in \mathcal{B}$  such that  $0 \notin \text{CH}(a_1, \dots, a_n) + \text{CH}(B)$ . Trying out all the tuples  $(a_1, a_2, \dots, a_n)$  is not tractable. Instead we use a separation result.

► **Theorem 10.** *Let  $\langle \mathcal{A}, \mathcal{B} \rangle$  be a Minkowski game with finite moves in  $\mathbb{R}^d$ , and let  $C = \{e_1, \dots, e_d\}$  be the canonical basis of  $\mathbb{R}^d$ . The game is won by Player B iff there exist  $B \in \mathcal{B}$  and affinely independent  $x_1, \dots, x_d \in (\cup \mathcal{A} + B) \cup C$  s.t. for all  $A \in \mathcal{A}$  there exists  $a \in A$  s.t. the affine hull of  $x_1, \dots, x_d$  separates  $a + B$  from 0.*

► **Corollary 11.** *Deciding the winner of a boundedness Minkowski game with rational polytopic moves in  $\mathbb{R}^d$  that involve  $n$  vertices can be done in time  $O(n^{2d+2})$ .*

**Moves defined in the first-order theory of the reals.** Here we show that if the moves are definable in the first-order theory of the reals, then so is the condition

$$0 \in (\text{CH}\{a_i \mid i \leq n\}) + \text{CH}(\overline{B}) \text{ for all } (a_i)_{i \leq n} \text{ with } a_i \in \text{CH}(\overline{A_i}) \text{ and } B \in \mathcal{B} \quad (1)$$

from Theorem 5. As the first-order theory is decidable, so is the question of who is winning a given boundedness Minkowski game with such moves.

We consider first-order formulas with binary function symbols  $+$  and  $\cdot$ , constants 0 and 1 and binary relation symbol  $<$ . A move  $A \subseteq \mathbb{R}^d$  is defined by some formula  $\phi_A$  with  $d$  free variables  $x_1, \dots, x_k$  iff  $A = \{(x_1, \dots, x_k) \in \mathbb{R}^d \mid \phi(x_1, \dots, x_n)\}$ . If  $\phi$  defines  $A$ , then

$$\begin{aligned} \phi_{\text{conv}} = & \exists a_1^1, \dots, a_{d+1}^1, \dots, a_{d+1}^d, \alpha_1, \dots, \alpha_{d+1} \bigwedge_{i=1}^{d+1} \phi(a_i^1, \dots, a_i^d) \\ & \wedge \sum_{i=1}^{d+1} \alpha_i = 1 \wedge \bigwedge_{i=1}^{d+1} 0 \leq \alpha_i \wedge \bigwedge_{j=1}^d \left( x_j = \sum_{i=1}^{d+1} \alpha_i a_i^j \right) \end{aligned}$$

defines  $\text{CH}(A)$ . Also, the formula

$$\phi_{\text{cl}} = \forall \varepsilon \quad \varepsilon > 0 \Rightarrow \left( \exists a_1, \dots, a_d \quad \phi(a_1, \dots, a_d) \wedge \bigwedge_{i=1}^d a_i < x_i + \varepsilon \wedge x_i < a_i + \varepsilon \right)$$

defines  $\overline{A}$ . It then follows that the condition (1) above is expressible as some formula  $\phi_{\text{win}}$  obtained from the formulas  $\phi_A, \phi_B$  defining the moves in  $\mathcal{A}$  and  $\mathcal{B}$ . Moreover, the length of the formula  $\phi_{\text{win}}$  is polynomially bounded in the sum of the length of the  $\phi_A, \phi_B$ .

► **Proposition 12.** Deciding the winner of a boundedness Minkowski game with moves defined in the first-order theory of the reals is 2EXPTIME-COMplete.

**Proof Sketch.** Since by [3, 5] deciding truth in the first-order theory of the reals is 2EXPTIME-COMplete. ◀

**The computable analysis perspective.** If we represent the sets involved in the boundedness Minkowski games via polyhedra or first order formula, we have only restricted expressivity available to us. Using notions from computable analysis [17], we can however consider computability for all boundedness Minkowski games with closed moves – which is not a problematic restriction. As the involved spaces are all connected, we cannot expect decidability, and instead turn our attention to semidecidability, i.e. truth values in the Sierpinski space  $\mathbb{S}$ .

We do have to decide on a representation for the sets, though. Pointing to [14] for definitions and explanations, we have the spaces  $\mathcal{A}(\mathbb{R}^d)$  of closed subsets,  $\mathcal{K}(\mathbb{R}^d)$  of compact subsets and  $\mathcal{V}(\mathbb{R}^d)$  of overt subsets available. In  $\mathcal{A}(\mathbb{R}^d)$ , a closed subset can be seen as being represented by an enumeration of rational balls exhausting its complement. The space  $\mathcal{K}(\mathbb{R}^d)$  adds to that some  $K \in \mathbb{N}$  such that the set is contained in  $[-K, K]^d$ . In  $\mathcal{V}(\mathbb{R}^d)$ , a closed subset is instead represented by listing all rational balls intersecting it.

A relevant property is that universal quantification over compact sets from  $\mathcal{K}(\mathbb{R}^d)$  and existential quantification over overt sets from  $\mathcal{V}(\mathbb{R}^d)$  preserve open predicates. We can use the former to find that:

► **Proposition 13.** The Minkowski sum  $+ : \mathcal{A}(\mathbb{R}^d) + \mathcal{K}(\mathbb{R}^d) \rightarrow \mathcal{A}(\mathbb{R}^d)$  is computable.

It was already shown in [10, Proposition 1.5] (also [18]) that convex hull is a computable operation on compact sets, but not on closed sets. Put together, we find that:

► **Proposition 14.** Consider boundedness Minkowski games, where moves  $A \in \mathcal{A}$  are given as overt sets (i.e. in  $\mathcal{V}(\mathbb{R}^d)$ ) and moves  $B \in \mathcal{B}$  are given as compact sets (i.e. in  $\mathcal{K}(\mathbb{R}^d)$ ). The set of games won by Player B constitutes a computable open subset.

## 5 The safety problem

We now turn our attention to the safety Minkowski games. We want to understand which initial positions  $a_0 \in \text{Safe}$  give Player A a winning strategy in the safety game  $\langle \mathcal{A}, \mathcal{B}, \text{Safe}, a_0 \rangle$ . In a minor abuse of notation, we speak of **the** safety Minkowski game  $\langle \mathcal{A}, \mathcal{B}, \text{Safe} \rangle$ , and call the set of  $a_0$  such that Player A has a winning strategy the *winning region*  $W$ .

We give two general results below: first, the winning region is the greatest fixed point of an operator that removes the points where Player B can provably win (in finitely many rounds); second, for finite  $\mathcal{A}$  this fixed point is approximated in a Kleene fixed-point style.

Let  $\langle \mathcal{A}, \mathcal{B}, \text{Safe} \rangle$  be a safety game. Given  $E$  a target set,  $f(E)$  is defined below as the positions from where Player A can ensure to fall in  $E$  after one round of the game.

► **Definition 15.** For all  $E \subseteq \mathbb{R}^d$  let  $g(E) := f(E) \cap \text{Safe}$ , where

$$f(E) := \{x \in \mathbb{R}^d \mid \exists A \in \mathcal{A}, \forall a \in A, \forall B \in \mathcal{B}, \exists b \in B, x + a + b \in E\}.$$

► **Lemma 16.** *The winning region  $W$  is the greatest fixed point of  $g$ .*

► **Proposition 17.** Let  $S_0 := \mathbb{R}^d$ , let  $S_{n+1} := g(S_n)$  for all  $n$ , and let  $S_\omega := \bigcap_{n \in \mathbb{N}} S_n$ .  $S_\omega$  is the greatest fixed point of  $g$ .

**Computable analysis perspective.** We start our investigation of the computational complexity of determining the winner in safety Minkowski games by considering the general setting of computable analysis, as we did in the end of Section 4 for the boundedness games. We point again to [14] for notation and definition, and in particular make use of the characterizations of  $\mathcal{V}(\mathbb{R}^d)$  and  $\mathcal{K}(\mathbb{R}^d)$  via the preservation of open predicates under quantification. We obtain:

► **Proposition 18.** Given a safety Minkowski game  $\langle \mathcal{A}, \mathcal{B}, \text{Safe}, a_0 \rangle$  with finite  $\mathcal{A}$  of overt sets (i.e. from  $\mathcal{V}(\mathbb{R}^d)$ ), finite  $\mathcal{B}$  of compact sets (i.e. from  $\mathcal{K}(\mathbb{R}^d)$ ), and closed  $\text{Safe}$  (i.e. from  $\mathcal{A}(\mathbb{R}^d)$ ), we can semidecide (recognize) if Player B has a winning strategy.

### Undecidability for polytopic sets

► **Theorem 19.** *There is  $d \in \mathbb{N}$ , a convex rational polytope  $\text{Safe}$  and a finite family  $\mathcal{A}$  of closed convex rational polytopes all in  $\mathbb{R}^d$  such that it is undecidable, whether Player A has a winning strategy in the one-sided safety Minkowski game  $\langle \mathcal{A}, \text{Safe}, a_0 \rangle$ , given  $a_0$  as a rational vector.*

To prove this theorem, we provide a reduction from the control state reachability problem for two counter machines to the problem of deciding if Player B has a winning strategy in a safety Minkowski game. As the first step, we introduce a slightly more general version of one-sided Minkowski games, and show a reduction to one-sided safety Minkowski games:

► **Definition 20.** A safety-reachability one-sided Minkowski game is given by a tuple  $\langle \mathcal{A}, \text{Safe}, \text{Goal}, a_0 \rangle$ , where  $\langle \mathcal{A}, \text{Safe}, a_0 \rangle$  is some  $d$ -dimensional safety one-sided Minkowski game, and  $\text{Goal} \subseteq \text{Safe}$ . It is played like the safety Minkowski game, and if Player A wins  $\langle \mathcal{A}, \text{Safe}, a_0 \rangle$ , then he wins  $\langle \mathcal{A}, \text{Safe}, \text{Goal}, a_0 \rangle$ . If the play enters  $\text{Goal}$  prior to leaving  $\text{Safe}$  for the first time, also Player A wins. Else Player B wins.

► **Proposition 21.** Given a  $d$ -dimensional safety-reachability one-sided Minkowski game  $\langle \mathcal{A}, \text{Safe}, \text{Goal}, a_0 \rangle$ , we define the associated  $d + 1$ -dimensional safety one-sided Minkowski game  $\langle \mathcal{A}', \text{Safe}', a'_0 \rangle$  as follows:

1.  $a'_0 := \langle a_0, 0 \rangle$
2.  $\text{Safe}' := \text{CH}((\text{Safe} \times \{0\}) \cup (\text{Goal} \times \{1\}))$
3.  $\mathcal{A}' := \{A \times \{0\} \mid A \in \mathcal{A}\} \cup \{(0, \dots, 0, 1)\}, \{(0, \dots, 0, -1)\}$

Now Player A (resp. Player B) has a winning strategy in the original game iff he (resp. she) has one in the associated game.

**2CM and the control state reachability problem.** A two-counter machine, 2CM for short, is defined by a finite directed graph  $(Q, E)$  with labeled edges. Vertices have out-degree 0, 1 or 2. If the out-degree is 1, the corresponding edge is labeled with one of  $\text{INC}^i$ ,  $\text{DEC}^i$  for  $i \in \{0, 1\}$ . If it is 2, one outgoing edge is labeled with  $\text{isZero}^i$  and the other with  $\text{isNotZero}^i$  for some  $i \in \{0, 1\}$ . There is a designated starting vertex  $q_0 \in Q$ .

A finite or infinite path through the graph is a *valid computation starting from  $n_0$  and  $n_1$*  if the following is true: the path starts at  $q_0$ . If one starts with  $c_0 := n_0$  and  $c_1 := n_1$  and increments (decrements)  $c_i$  by 1 whenever encountering a label  $\text{INC}^i$  ( $\text{DEC}^i$ ), then at the moment an edge labeled with  $\text{isZero}^i$  ( $\text{isNotZero}^i$ ) is passed, we find that  $c_i = 0$  ( $c_i \neq 0$ ).

► **Theorem 22** ([12, Theorem 1a]). *There is a 2CM such that it is undecidable whether there exists an infinite valid computation starting from  $n_0$  and  $n_1$  (where  $n_0$  and  $n_1$  are the input).*

We will slightly modify the 2CM to simplify the construction. We subdivide every edge by adding another vertex on it. If the original edge was labeled  $\text{INC}^i$  ( $\text{DEC}^i$ ), then the two new edges will be labeled  $\text{INCa}^i$  and  $\text{INCb}^i$  ( $\text{DECa}^i$  and  $\text{DECb}^i$ ). If the original edge was labeled  $\text{isZero}^i$  or  $\text{isNotZero}^i$ , we move the label to the newly-added vertex.

Now we are ready to reduce the non-halting problem of modified 2CM's to the existence of a winning strategy for Player A in a safety-reachability one-sided Minkowski game. The general idea of the reduction is as follows. First, Player A is forced to simulate the computation of the 2CM in order to avoid violating the safety condition of the safety Minkowski game.



The value of each counter  $c_i$ ,  $i \in \{1, 2\}$ , is coded in some dimension  $y_i$  such that when the counter  $c_i$  is equal to  $k \in \mathbb{N}$  then the value of  $y_i = \frac{1}{2^k}$ . The role of **Player B** is restricted to assist **Player A** to multiply or divide the  $x_i$  by 2. His failure to operate as intended will let the play reach **Goal**. Additionally, each vertex  $Q$  is associated with one dimension that will be non-zero iff the computation is currently in that vertex.

All the moves and invariants that we use are definable by finite sets of linear constraints.

**Defining the reduction.** We are given a modified 2CM with vertex set  $Q$  (called control states) and edges  $E$ . The associated safety-reachability Minkowski game will be played in  $\mathbb{R}^{4+|Q|}$ . The first 4 dimensions are  $(x_0, y_0, x_1, y_1)$ , where the  $y_i$  encode the counter values, and the  $x_i$  are auxiliary values. The remaining  $|Q|$  dimensions are indexed with the states  $q$ .

Every instruction  $e \in E$  corresponds to some move  $A_e$  for **Player A**. The move  $A_e$  will always decompose as  $A_e = A_e^{xy} \times \{a_e^Q\}$ . If  $e$  is an edge from  $q_i$  to  $q_f$ , then  $a_e^Q \in \mathbb{R}^{|Q|}$  will have  $-1$  at component  $q_i$ ,  $+1$  at component  $q_f$  and 0 elsewhere.

Label of $e$	Value of $A_e^{xy}$	Label of $e$	Value of $A_e^{xy}$
-	$\{(0, 0, 0, 0)\}$		
INCa <sup>0</sup>	$\text{CH}\{(0, 0), (1, -1)\} \times \{(0, 0)\}$	DECa <sup>0</sup>	$\text{CH}\{(0, 0), (1, 0)\} \times \{(0, 0)\}$
INCa <sup>1</sup>	$\{(0, 0)\} \times \text{CH}\{(0, 0), (1, -1)\}$	DECa <sup>1</sup>	$\{(0, 0)\} \times \text{CH}\{(0, 0), (1, 0)\}$
INCb <sup>0</sup>	$\text{CH}\{(0, 0), (-1, 0)\} \times \{0, 0\}$	DECb <sup>0</sup>	$\text{CH}\{(0, 0), (-1, 1)\} \times \{(0, 0)\}$
INCb <sup>1</sup>	$\{0, 0\} \times \text{CH}\{(0, 0), (-1, 0)\}$	DECb <sup>1</sup>	$\{0, 0\} \times \text{CH}\{(0, 0), (-1, 1)\}$

It remains to define the sets **Safe** and **Goal**. For that, let  $Q_z^i$  be the set of states labeled with  $\text{isZero}^?^i$ , and let  $Q_n^i$  be the set of states labeled with  $\text{isNotZero}^?^i$ . Let  $Q_o$  be the set of unlabeled states with non-zero outdegree. Let  $e_q$  be the  $|Q|$ -dimensional vector having 1 in component  $q$  and 0 elsewhere.

$$\begin{aligned} \text{Safe} := & \text{CH} \left[ \left( \bigcup_{q \in Q_o} [0, 1]^4 \times \{e_q\} \right) \cup \left( \bigcup_{q \in Q_n^0} [0, 1] \times [0, 0.7] \times [0, 1]^2 \times \{e_q\} \right) \right. \\ & \cup \left( \bigcup_{q \in Q_n^1} [0, 1]^3 \times [0, 0.7] \times \{e_q\} \right) \cup \left( \bigcup_{q \in Q_z^0} [0, 1] \times \{1\} \times [0, 1]^2 \times \{e_q\} \right) \\ & \left. \cup \left( \bigcup_{q \in Q_z^1} [0, 1]^3 \times \{1\} \times \{e_q\} \right) \right] \end{aligned}$$

$$\text{Goal} := \text{Safe} \cap (\{(x, y) \in \mathbb{R}^2 \mid y \neq x \neq 0\} \times \mathbb{R}^{2+|Q|} \cup \mathbb{R}^2 \times \{(x, y) \in \mathbb{R}^2 \mid y \neq x \neq 0\} \times \mathbb{R}^{|Q|})$$

The starting position of the game is as follows:  $(0, 2^{-n_0}, 0, 2^{-n_1}, 0, \dots, 0, 1, 0, \dots)$ , where  $n_0$  and  $n_1$  are the starting values for the counters, and the unique 1 in the latter part is found at the index corresponding to the starting state of the 2CM.

**Correctness of the reduction.** We claim that **Player A** has a winning strategy in the constructed game, iff the (modified) 2CM has a valid infinite computation path. As moves correspond to edges, every sequence of moves chosen by **Player A** in the game can be seen as a sequence of edges for the 2CM.

First we argue that every sequence of edges which is not a path induces a losing strategy in the game. As the values of the components associated with the control states must remain between 0 and 1, and every move has components  $-1$ ,  $+1$  somewhere and 0 elsewhere it

follows that every non-losing sequences of moves ensure that exactly one state-component  $q_i$  of the position is 1, and the others are 0. Every move coming from an edge not having the initial state  $q_i$  will lose immediately.

Next, we shall explain how the moves for  $\text{INCa}^i$  and  $\text{INCb}^i$  together cause the desired effect. If the current relevant part of the position is  $(0, 2^{-k})$ , then after the move  $\text{INCa}^i$  Player B may pick any  $(x, y) \in (0, 2^{-k}) + \text{CH}\{(0, 0), (1, -1)\}$ , in other words, Player B picks some  $t \in [0, 1]$  and sets the position to  $(t, 2^{-k} - t)$ . If Player B picks  $t = 0$ , then Player A can repeat the same move. By the definition of Goal, the only other safe choice for Player B is to pick  $t = 2^{-k-1}$ , i.e. to set the position to  $(2^{-k-1}, 2^{-k-1})$ . The move associated with  $\text{INCb}^i$  follows, which means that Player B gets to pick some  $(2^{-k-1} - t, 2^{-k-1})$ . Again, choosing  $t = 0$  lets Player A repeat her move, and the only other choice compatible with avoiding Goal is to move to  $(0, 2^{-k-1})$ .

The construction for  $\text{DECa}^i$ ,  $\text{DECb}^i$  is similar: starting at  $(0, 2^{-k})$ , Player B can only remain, enter Goal or move to  $(2^{-k}, 2^{-k})$  if Player A plays a move corresponding to  $\text{DECa}^i$ . The subsequent  $\text{DECb}^i$  move allows Player B to remain, enter Goal or to move to  $(0, 2^{-k+1})$ .

Finally, we need to discuss (conditional) halting: by the construction of Safe, if a vertex with out-degree 0 is reached, or a vertex labeled with an unsatisfied condition, then the play is loosing for Player A. Thus, winning strategies of Player A correspond exactly to infinite non-halting computations of the 2CM.

**Structural safety games.** The above undecidability result for safety game with polytopic sets motivates the study of *structural safety Minkowski games*. In a (one-sided) structural safety game, there is no designated initial state and Player A is asked to be able to maintain the system safe starting from any point in the safe region. It is not difficult to see that this stronger requirement makes the game equivalent to a "one round" game. Indeed, if Player A can maintain safety from all positions within Safe, then it means that after one round of the game, the game is again within Safe, from which Player A can win for one more round, etc.

The complexity of the structural safety games for polytopic moves and Safe is below.

► **Theorem 23.** *Given a one-sided structural safety Minkowski game  $\langle \mathcal{A}, \mathcal{B}, \text{Safe} \rangle$  where moves and the set Safe are rational polytopic, it is CONP-COMPLETE to decide if Player A has a winning strategy from all positions in Safe.*

## 6 Open questions

By comparing the results from Section 4 on linear constraints and fixed dimension, we see that while deciding the winner in a boundedness Minkowski game is coNP-complete in general, it becomes polynomial-time if the dimension of the ambient space is fixed. Thus, it makes a good candidate for an investigation in the setting of parameterized complexity [6]. Is the problem fixed-parameter tractable? Is it hard for some  $\text{W}[n]$ -class?

We showed that from some dimension onwards, it is undecidable to determine the winner in a safety Minkowski game defined via sets of linear constraints defining open and closed convex polytopes. This immediately raises two questions: first, what happens for small dimensions? Given that our construction uses essentially two dimensions per instruction, and two per counter, an optimal value is presumably obtained by using universal machine having more than 2 counters. Second, what happens if we restrict our attention to games defined via sets of linear constraints that are all non strict (defining closed convex polytopes only)?

**Acknowledgements.** We would like to thank Dr. Pritha Mahata and Prof. Jean Cardinal for helpful and stimulating discussions related to a.o. convex polyhedra and linear algebra.

---

**References**

---

- 1 Rajeev Alur, Vojtech Forejt, Salar Moarref, and Ashutosh Trivedi. Safe schedulability of bounded-rate multi-mode systems. In *Proceedings of the 16th international conference on Hybrid systems: computation and control, HSCC 2013, April 8-11, 2013, Philadelphia, PA, USA*, pages 243–252. ACM, 2013.
- 2 Rajeev Alur, Ashutosh Trivedi, and Dominik Wojtczak. Optimal scheduling for constant-rate multi-mode systems. In *Hybrid Systems: Computation and Control (part of CPS Week 2012), HSCC'12, Beijing, China, April 17-19, 2012*, pages 75–84. ACM, 2012.
- 3 Michael Ben-Or, Dexter Kozen, and John Reif. The complexity of elementary algebra and geometry. *Journal of Computer and Systems Sciences*, 32(2):251–264, 1986. doi:10.1016/0022-0000(86)90029-2.
- 4 Bernard Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10:377–409, 1993. doi:10.1007/BF02573985.
- 5 James H. Davenport and Joos Heintz. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, 5(1):29–35, 1988. doi:10.1016/S0747-7171(88)80004-X.
- 6 Rod Downey and Michael Fellows. *Parameterized Complexity*. Springer, 1999. doi:10.1007/978-1-4612-0515-9.
- 7 Laurent Doyen and Alexander Rabinovich. Robot games. Research Report LSV-13-02, Laboratoire Spécification et Vérification, ENS Cachan, France, 2013. URL: [http://www.lsv.ens-cachan.fr/Publis/RAPPORTS\\_LSV/PDF/rr-lsv-2013-02.pdf](http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/PDF/rr-lsv-2013-02.pdf).
- 8 Thomas A. Henzinger, Benjamin Horowitz, and Rupak Majumdar. Rectangular hybrid games. In *CONCUR'99: Concurrency Theory, 10th International Conference, Eindhoven, The Netherlands, August 24-27, 1999, Proceedings*, volume 1664 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 1999.
- 9 Thomas A. Henzinger and Peter W. Kopke. Discrete-time control for rectangular hybrid automata. In *Automata, Languages and Programming, 24th International Colloquium, ICALP'97, Bologna, Italy, 7-11 July 1997, Proceedings*, volume 1256 of *Lecture Notes in Computer Science*, pages 582–593. Springer, 1997.
- 10 Stéphane Le Roux and Arno Pauly. Finite choice, convex choice and finding roots. *Logical Methods in Computer Science*, 2015. doi:10.2168/LMCS-11(4:6)2015.
- 11 Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In *STACS*, pages 229–242, 1995.
- 12 Marvin L. Minsky. Recursive unsolvability of Post's problem of “tag” and other topics in theory of Turing machines. *Annals of Mathematics*, 74(3):437–455, 1961.
- 13 Reino Niskanen, Igor Potapov, and Julien Reichert. Undecidability of Two-dimensional Robot Games. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *41st Int. Sym. on Mathematical Foundations of Computer Science (MFCS 2016)*, volume 58 of *LIPICs*, pages 73:1–73:13. Schloss Dagstuhl, 2016. doi:10.4230/LIPICs.MFCS.2016.73.
- 14 Arno Pauly. On the topological aspects of the theory of represented spaces. *Computability*, 5(2):159–180, 2016. doi:10.3233/COM-150049.
- 15 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 179–190. ACM Press, 1989.
- 16 Stéphane Le Roux, Arno Pauly, and Jean-François Raskin. Minkowski games. *CoRR*, abs/1609.07048, 2016. URL: <http://arxiv.org/abs/1609.07048>.
- 17 Klaus Weihrauch. *Computable Analysis*. Springer-Verlag, 2000.
- 18 Martin Ziegler. Computable operators on regular sets. *Mathematical Logic Quarterly*, 50:392–404, 2004.



# On the Sensitivity Complexity of $k$ -Uniform Hypergraph Properties\*

Qian Li<sup>1</sup> and Xiaoming Sun<sup>2</sup>

- 1 CAS Key Lab of Network Data Science and Technology, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China; and University of Chinese Academy of Sciences, Beijing, China  
liqian@ict.ac.cn
- 2 CAS Key Lab of Network Data Science and Technology, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China; and University of Chinese Academy of Sciences, Beijing, China  
sunxiaoming@ict.ac.cn

---

## Abstract

In this paper we investigate the sensitivity complexity of hypergraph properties. We present a  $k$ -uniform hypergraph property with sensitivity complexity  $O(n^{\lceil k/3 \rceil})$  for any  $k \geq 3$ , where  $n$  is the number of vertices. Moreover, we can do better when  $k \equiv 1 \pmod{3}$  by presenting a  $k$ -uniform hypergraph property with sensitivity  $O(n^{\lceil k/3 \rceil - 1/2})$ . This result disproves a conjecture of Babai [9], which conjectures that the sensitivity complexity of  $k$ -uniform hypergraph properties is at least  $\Omega(n^{k/2})$ . We also investigate the sensitivity complexity of other weakly symmetric functions and show that for many classes of transitive-invariant Boolean functions the minimum achievable sensitivity complexity can be  $O(N^{1/3})$ , where  $N$  is the number of variables. Finally, we give a lower bound for sensitivity of  $k$ -uniform hypergraph properties, which implies the *sensitivity conjecture* of  $k$ -uniform hypergraph properties for any constant  $k$ .

**1998 ACM Subject Classification** F.1.3 Complexity Measures and Classes

**Keywords and phrases** Sensitivity Complexity,  $k$ -uniform Hypergraph Properties, Boolean Function, Turan's question

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.51

## 1 Introduction

In order to understand the effect of symmetry on computational complexity, especially in the decision tree model, Boolean functions with certain symmetry have been extensively investigated. It is observed that symmetry usually implies high complexity or makes the problem harder in the decision tree model. An illustrative example is the well known *evasiveness conjecture*, which asserts that any non-constant monotone transitive Boolean function is evasive, and it has attracted a lot of attention [29, 13, 25, 6]. Rivest and Vuillemin [32] showed that any non-constant monotone graph property is weakly evasive. Kulkarni et al. [26] showed an analogous result for 3-hypergraph properties. Black [10] extended these results to  $k$ -uniform hypergraph properties for any fixed  $k$ .

Sensitivity complexity is an important complexity measure of Boolean functions in the decision tree model, and sensitivity complexity of Boolean functions with certain symmetry

---

\* This work was supported in part by the National Natural Science Foundation of China Grant 61222202, 61433014, 61502449, 61602440, the 973 Program of China Grants No. 2016YFB1000201 and the China National Program for support of Top-notch Young Professionals.



has also attracted a lot of attention. One of the most challenging problem here is whether *symmetry* implies high sensitivity complexity. The famous sensitivity conjecture, which asserts sensitivity complexity and block sensitivity are polynomially related, implies  $s(f) = \Omega(n^\alpha)$  for transitive functions with some constant  $\alpha > 0$  since it has been shown that  $bs(f) = \Omega(n^{1/3})$  for transitive functions [34]. Turan [38] initiated the study of sensitivity of graph properties and proved that the sensitivity is greater than  $n/4$  for any nontrivial graph property, where  $n$  is the number of vertices, and this relation is also tight up to a constant factor. He also pointed out that for symmetric functions,  $s(f) \geq n/2 \geq bs(f)/2$ . Recently the lower bound has been improved to  $\frac{6}{17}n$  by Sun [35], and further been improved to  $\lfloor \frac{n}{2} \rfloor$  for sufficient large  $n$  by Karpas [24]. Gao et al. [18] investigated the sensitivity of bipartite graph properties as well. In 2005, Chakraborty [14] constructed a minterm cyclically invariant Boolean function whose sensitivity is  $\Theta(n^{1/3})$ , which answers Turan's question [38] in the negative. He also showed this bound is tight for minterm transitive functions.

For hypergraph properties, Biderman et al. [9] present a sequence of  $k$ -uniform hypergraph properties with sensitivity  $\Theta(\sqrt{N})$ , where  $N = \binom{n}{k}$  is the number of variables. Babai conjectures that this bound is tight, i.e.,  $s(f) = \Omega(\sqrt{N})$  for any nontrivial  $k$ -uniform hypergraph property  $f$ .

**Our Results.** In this paper we disprove this conjecture by constructing  $k$ -uniform hypergraph properties with sensitivity  $O(n^{\lceil k/3 \rceil})$ .

► **Theorem 1.** *For any fixed  $k \geq 3$ , there exists a sequence of  $k$ -uniform hypergraph properties  $f$  such that  $s(f) = O(n^{\lceil k/3 \rceil})$ , where  $n$  is the number of vertices.*

Moreover, we can give better constructions when  $k \equiv 1 \pmod{3}$ .

► **Theorem 2.** *For any fixed  $k \geq 4$  satisfying  $k \equiv 1 \pmod{3}$ , there exists a sequence of  $k$ -uniform hypergraph properties  $f$  such that  $s(f) = O(n^{\lceil k/3 \rceil - 1/2})$ , where  $n$  is the number of vertices.*

More generally, we also investigate the sensitivity of  $k$ -partite  $k$ -uniform hypergraph properties. Actually, the constructions of  $k$ -uniform hypergraph properties are inspired by the constructions of  $k$ -partite  $k$ -uniform hypergraph properties.

► **Theorem 3.** *For any  $k \geq 3$ , there exists a sequence of  $k$ -partite  $k$ -uniform hypergraph properties  $f : \{0, 1\}^{n^k} \rightarrow \{0, 1\}$  such that  $s(f) = O(n^{\lceil k/3 \rceil})$ .*

► **Theorem 4.** *For any  $k \geq 4$  satisfying  $k \equiv 1 \pmod{3}$ , there exists a sequence of  $k$ -partite  $k$ -uniform hypergraph properties  $f : \{0, 1\}^{n^k} \rightarrow \{0, 1\}$  such that  $s(f) = O(n^{\lceil k/3 \rceil - 1/2})$ .*

Let  $G$  be an Abelian group, the fundamental theorem of finite abelian groups states that  $G \cong C_{m_1} \times \cdots \times C_{m_l}$ , where  $C_m$  is the cyclic group of order  $m$  and  $|G| = \prod_{i=1}^l m_i$ .

► **Theorem 5.** *Let  $G \leq S_n$  be a transitive Abelian group, then there exists a Boolean function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  invariant under  $G$  such that  $s(f) \leq \alpha n^{1/3}$ , where  $\alpha$  is a number only depending on  $l$ .*

On the other side, Chakraborty [15] observed the following lower bound on  $k$ -uniform hypergraph properties without showing the proof, which implies the sensitivity conjecture of  $k$ -uniform hypergraph properties for any constant  $k$ . For the convenience of readers, the proof will be given in the paper.

► **Theorem 6.** *For any fixed  $k$  and any non-trivial  $k$ -uniform hypergraph property  $f$ ,  $s(f) = \Omega(n)$ , where  $n$  is the number of vertices.*

Similar lower bound holds for the sensitivity of  $k$ -partite  $k$ -uniform hypergraph properties.

► **Theorem 7.** *For any fixed  $k$  and any non-trivial  $k$ -partite  $k$ -uniform hypergraph property  $f$ ,  $s(f) = \Omega(n)$ , where  $n$  is the number of vertices in one partition.*

The sketch of the proof is as follows: we just pretend this function is a bipartite graph property by dividing  $k$  partitions into two sets of size 1 and  $k - 1$  respectively and then follow the same argument in the proof of Theorem 2 in [18]. We omit the proof in this paper.

**Related Work.** Sensitivity complexity and block sensitivity are first introduced by Cook, Dwork and Reischuk [16, 17] and Nisan [30] respectively, to study the time complexity of CREW-PRAMs. Block sensitivity has been shown to be polynomially related to a number of other complexity measures [12], such as decision tree complexity, certificate complexity, polynomial degree and quantum query complexity, etc, except sensitivity. The famous sensitivity conjecture, proposed by Nisan and Szegedy [31], asserts that block sensitivity and sensitivity complexity are also polynomially related. On one side, it is easy to see  $s(f) \leq bs(f)$  for any Boolean function  $f$  according to the definitions. On the other side, it is much more challenging to prove or disprove that block sensitivity is polynomially bounded by sensitivity. Despite of a lot of effort, the best known upper bound is exponential:  $bs(f) \leq \max\{2^{s(f)-1}(s(f) - \frac{1}{3}), s(f)\}$  [3]. Recently, He, Li and Sun further improve the upper bound to  $(\frac{8}{9} + o(1))s(f)2^{s(f)-1}$  [23]. The best known separation between sensitivity and block sensitivity is quadratic [4]: there exists a sequence of Boolean functions  $f$  with  $bs(f) = \frac{2}{3}s(f)^2 - \frac{1}{3}s(f)$ . For an excellent survey on the sensitivity conjecture, see [22]. For other recent progress, see [11, 2, 1, 5, 19, 36, 20, 21, 37, 8, 28, 33, 7].

**Organization.** We present some preliminaries in Section 2, and give the proofs of Theorem 1 and Theorem 2 in Section 3. We give the constructions of  $k$ -partite  $k$ -uniform hypergraph properties (Theorem 3 and 4) and the proof of Theorem 5 in Section 4 and give the proof of Theorem 6 in Section 5. Finally, we conclude this paper with some open problems in Section 6.

## 2 Preliminaries

Let  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  be a Boolean function and  $[n] = \{1, 2, \dots, n\}$ . For an input  $x \in \{0, 1\}^n$  and a subset  $B \subseteq [n]$ ,  $x^B$  denotes the input obtained by flipping all the bit  $x_j$  such that  $j \in B$ .

► **Definition 8.** The *sensitivity* of  $f$  on input  $x$  is defined as  $s(f, x) := |\{i | f(x) \neq f(x^{\{i\}})\}|$ . The sensitivity, 0-sensitivity and 1-sensitivity of the function  $f$  are defined as  $s(f) := \max_x s(f, x)$ ,  $s_0(f) = \max_{x \in f^{-1}(0)} s(f, x)$  and  $s_1(f) = \max_{x \in f^{-1}(1)} s(f, x)$  respectively.

► **Definition 9.** The *block sensitivity*  $bs(f, x)$  of  $f$  on input  $x$  is the maximum number of disjoint subsets  $B_1, B_2, \dots, B_r$  of  $[n]$  such that for all  $j \in [r]$ ,  $f(x) \neq f(x^{B_j})$ . The block sensitivity of  $f$  is defined as  $bs(f) = \max_x bs(f, x)$ .

► **Definition 10.** A *partial assignment* is a function  $p : [n] \rightarrow \{0, 1, \star\}$ . We call  $S = \{i | p_i \neq \star\}$  the support of this partial assignment. We define the size of  $p$  (denoted by  $|p|$ ) to be  $|S|$ . We call  $x$  a (full) assignment if  $x : [n] \rightarrow \{0, 1\}$ . We say  $x$  is consistent with  $p$  if  $x|_S = p$ , i.e.,  $x_i = p_i$  for all  $i \in S$ .<sup>1</sup>

<sup>1</sup> The function  $p$  can be viewed as a vector, and we sometimes use  $p_i$  to represent  $p(i)$ .

► **Definition 11.** For  $b \in \{0, 1\}$ , a  $b$ -certificate for  $f$  is a partial assignment  $p$  such that  $f(x) = b$  whenever  $x$  is consistent with  $p$ .

The *certificate complexity*  $C(f, x)$  of  $f$  on input  $x$  is the minimum size of  $f(x)$ -certificate that is consistent with  $x$ . The certificate complexity of  $f$  is  $C(f) = \max_x C(f, x)$ .

The *1-certificate complexity* of  $f$  is  $C_1(f) = \max_{x \in f^{-1}(1)} C(f, x)$ , and similarly we define  $C_0(f)$ .

According to the definitions, it's easy to see  $s(f) \leq bs(f) \leq C(f)$ ,  $s_0(f) \leq C_0(f)$  and  $s_1(f) \leq C_1(f)$ .

► **Definition 12.** Let  $p$  and  $p'$  be two partial assignments, the distance between  $p$  and  $p'$  is defined as  $dist(p, p') = |\{i | p_i = 1 \text{ and } p'_i = 0, \text{ or } p_i = 0 \text{ and } p'_i = 1\}|$ .

► **Definition 13.** Let  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  be a Boolean function and  $G$  be a subgroup of  $S_n$ , we say that  $f$  is invariant under  $G$  if  $f(x_1, \dots, x_n) = f(x_{\sigma(1)}, \dots, x_{\sigma(n)})$  for any  $x \in \{0, 1\}^n$  and any  $\sigma \in G$ .

A Boolean function  $f$  is called *transitive* (or *weakly symmetric*) if  $G$  is a transitive group<sup>2</sup>. A Boolean function  $f$  is called *symmetric* if  $G = S_n$ .

► **Definition 14.** A Boolean function  $f$  invariant under a transitive group  $G$  is called *minterm-transitive* if there exists a partial assignment  $p$  such that  $f(x) = 1$  if and only if  $x$  is consistent with  $p^\sigma := (p_{\sigma(1)}, p_{\sigma(2)}, \dots, p_{\sigma(n)})$  for some  $\sigma \in G$ . We call  $p$  the *minterm* of  $f$ .

A Boolean string can represent a graph in the following manner:  $x_{(i,j)} = 1$  means there is an edge connecting vertex  $i$  and vertex  $j$ , and  $x_{i,j} = 0$  means there is no such edge. Graph properties are functions for which the value is independent with the labeling of vertices, i.e. two isomorphic graphs have the same function value.

► **Definition 15.** A Boolean function  $f : \{0, 1\}^{\binom{n}{2}} \rightarrow \{0, 1\}$  is called a *graph property* if for every input  $x = (x_{(1,2)}, \dots, x_{(n-1,n)})$  and every permutation  $\sigma \in S_n$ ,

$$f(x_{(1,2)}, \dots, x_{(n-1,n)}) = f(x_{(\sigma(1),\sigma(2))}, \dots, x_{(\sigma(n-1),\sigma(n))}).$$

Similarly, we define  $k$ -uniform hypergraph properties.

► **Definition 16.** A Boolean function  $f : \{0, 1\}^{\binom{n}{k}} \rightarrow \{0, 1\}$  is called a  $k$ -uniform hypergraph property if for every input  $x = (x_{(1,2,\dots,k)}, \dots, x_{(n-k+1,\dots,n-1,n)})$  and every permutation  $\sigma \in S_n$ ,

$$f(x_{(1,2,\dots,k)}, \dots, x_{(n-k+1,\dots,n-1,n)}) = f(x_{(\sigma(1),\sigma(2),\dots,\sigma(k))}, \dots, x_{(\sigma(n-k+1),\dots,\sigma(n-1),\sigma(n))}).$$

Let  $p$  be a partial assignment and  $\sigma \in S_n$ , we define  $\sigma(p)$  the induced shift of  $p$  by  $\sigma$ , i.e.,  $\sigma(p)_S = p_{\sigma(S)}$  for any subset of  $[n]$  of size  $k$   $S$ . Here  $\sigma(S) = \{\sigma(i) | i \in S\}$ .

► **Definition 17.** A Boolean function  $f : \{0, 1\}^{n^k} \rightarrow \{0, 1\}$  is called  $k$ -partite  $k$ -uniform hypergraph property, if for every input  $x = (x_{(1,1,\dots,1)}, \dots, x_{(n,n,\dots,n)})$  and every  $\sigma = (\sigma_1, \dots, \sigma_k) \in S_n^{\otimes k}$ ,

$$f(x_{(1,1,\dots,1)}, \dots, x_{(n,n,\dots,n)}) = f(x_{(\sigma_1(1),\dots,\sigma_k(1))}, \dots, x_{(\sigma_1(n),\dots,\sigma_k(n))}).$$

It is easy to see that any ( $k$ -partite)  $k$ -uniform hypergraph property is transitive.

<sup>2</sup> A group  $G \leq S_n$  is transitive if for every  $i < j$ , there exists a  $\sigma \in G$  such that  $\sigma(i) = j$ .



### 3 k-Uniform Hypergraph Properties

In this section, we give the proofs of Theorem 1 and Theorem 2.

► **Theorem 1** (restated). *For any fixed  $k \geq 3$ , there exists a sequence of  $k$ -uniform hypergraph properties  $f$  such that  $s(f) = O(n^{\lceil k/3 \rceil})$ , where  $n$  is the number of vertices.*

**Proof.** The function we construct is a minterm function. Let  $p$  be the minterm defining  $f$ , and it is constructed as follows:

First, let  $k_1$  and  $k_2$  be two integers such that  $k_1 + 2k_2 = k$  and  $\lfloor k/3 \rfloor \leq k_1, k_2 \leq \lceil k/3 \rceil$ . Let  $V = \{v_1, \dots, v_n\}$  be the set of vertices and  $B = \{v_n, v_{n-1}, \dots, v_{n-k_1+1}\}$ . For each  $1 \leq i \leq 6$ , let  $W_i = \{v_{(i-1)k_2+1}, \dots, v_{ik_2}\}$ , and  $C = \bigcup_{1 \leq i \leq 6} W_i$ ,  $D = V \setminus (C \cup B)$ .

- For any  $S \subseteq C$  of size  $2k_2$ ,  $p(B \cup S) = 0$ , except  $S = W_i \cup W_{i+1}$  for  $i \in [5]$  where  $p(B \cup S) = 1$ .
- For any  $S$  of size  $2k_2$  and  $k_2 \leq |S \cap C| < 2k_2$ ,  $p(B \cup S) = 1$ , except  $W_3$  or  $W_4 \subseteq S$  where  $p(B \cup S) = 0$ .
- All the other variables are  $\star$ .

If  $f(x) = 1$  then  $x$  is consistent with some  $\sigma(p)$ , which implies  $C(f, x) \leq |p|$ . Thus  $s_1(f) \leq C_1(f) \leq |p| = \sum_{i=k_2}^{2k_2} \binom{6k_2}{i} \binom{n-6k_2-k_1}{2k_2-i} = O(n^{k_2})$ . If  $f(x) = 0$ , then  $s(f, x)$  is at most the number of shifts of  $p$  (i.e.,  $\sigma(p)$ s) adjacent to  $x$ , thus according to the triangle inequality,  $s_0(f)$  is at most the maximum number of  $\sigma(p)$ s where the distance between any two of them is at most 2. We claim that for any shift  $\pi(p)$ , there are only  $O(1)$   $\sigma(p)$ s satisfying  $\pi(B) = \sigma(B)$  and  $\text{dist}(\pi(p), \sigma(p)) \leq 2$ . It is easy to see that this claim implies  $s_0(f) = O(n^{k_1})$  since there are  $\binom{n}{k_1} = O(n^{k_1})$  possible choices of the  $\sigma(B)$ s, and this will end the whole proof.

► **Claim 18.** *For any  $\pi(p)$ , there are only  $O(1)$   $\sigma(p)$ s satisfying  $\pi(B) = \sigma(B)$  and  $\text{dist}(\pi(p), \sigma(p)) \leq 2$ .*

*Proof.* It is easy to see that this claim is equivalent to show  $|\{\sigma(p) | \text{dist}(p, \sigma(p)) \leq 2 \text{ and } \sigma(B) = B\}| = O(1)$ . The case for  $k_2 = 1$  is a little special, and we discuss this case first.

**Case for  $k_2 = 1$ .** We use Figure 1 to illustrate  $p$ . Note that the vertices in  $D$  are symmetric and  $|C| = O(1)$ , thus  $|\{\sigma(p) | \sigma(C) = C \text{ and } \sigma(B) = B\}| = O(1)$ . So we only need to consider the set  $\{\sigma(p) | \sigma(C) \neq C \text{ and } \sigma(B) = B\}$ , and we exclude each  $\sigma$  case by case:

1.  $\sigma(W_3)$  or  $\sigma(W_4) \in \{W_1, W_2, W_5, W_6\}$ .

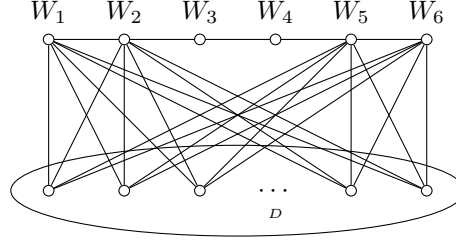
W.l.o.g, assume  $\sigma(W_3) = W_1$ , then

$$\begin{aligned}
 \text{dist}(p, \sigma(p)) &\geq |\{e \subseteq [n] | \sigma(p)(e) = 1, p(e) = 0, |e| = k, \{W_3, B\} \subseteq e\}| \\
 &\geq |\{e \subseteq [n] | \sigma(p)(e) = 1, |e| = k, \{W_3, B\} \subseteq e\}| \\
 &\quad - |\{e \subseteq [n] | p(e) = \{1, \star\}, |e| = k, \{W_3, B\} \subseteq e\}| \\
 &= |\{e \subseteq [n] | p(\sigma(e)) = 1, |e| = k, \{W_3, B\} \subseteq e\}| - O(1) \\
 &= |\{e \subseteq [n] | p(e) = 1, |e| = k, \{W_1, B\} \subseteq e\}| - O(1) \\
 &\geq n - O(1) \geq 3.
 \end{aligned}$$

2.  $\sigma(W_3)$  or  $\sigma(W_4) \in D$ , and  $\{\sigma(W_3), \sigma(W_4)\} \cap \{W_1, W_2, W_5, W_6\} = \emptyset$ .

W.l.o.g, assume  $\sigma(W_3) \in D$ , note that for any  $v$ ,  $p(B \cup W_3 \cup v) \neq \star$ , and  $|\{v \neq W_4 | p(B \cup W_3 \cup v) = 1\}| = 1$ . While  $|\{v \neq W_4 | \sigma(p)(B \cup W_3 \cup v) = 1\}| = |\{v \neq W_4 | p(B \cup \sigma(W_3) \cup \sigma(v)) = 1\}| = 4$ , thus  $\text{dist}(p, \sigma(p)) \geq 3$ .

3.  $\sigma(W_3) = W_3$  and  $\sigma(W_4) = W_4$ .



■ **Figure 1** The graph to illustrate  $p$  for  $k_2 = 1$ .

- a.  $\sigma(W_5) \neq W_5$  and  $\sigma(W_2) \neq W_2$ .  
 Since  $p(B \cup \sigma(W_2) \cup \sigma(W_3)) = p(B \cup \sigma(W_4) \cup \sigma(W_5)) = 0$  and  $p(B \cup \sigma(W_3) \cup \sigma(S)) = p(B \cup \sigma(W_4) \cup \sigma(S')) = 1$ , for some  $\sigma(S) = W_2$  and  $\sigma(S') = W_5$ , thus  $\text{dist}(p, \sigma(p)) \geq 4$ .
- b.  $\sigma(W_5) = W_5$  and  $\sigma(W_2) = W_2$ .  
 Since  $\sigma(C) \neq C$ , W.l.o.g, assume  $\sigma(W_1) \in D$ , then  $p(B \cup \sigma(W_1) \cup \sigma(W_5)) = 1$ .  
 If  $\sigma(W_6) \in D$ , then  $p(B \cup \sigma(W_2) \cup \sigma(W_6)) = 1$  and  $p(B \cup \sigma(S) \cup \sigma(W_5)) = p(B \cup \sigma(S') \cup \sigma(W_2)) = 0$ , for some  $\sigma(S) = W_1$  and  $\sigma(S') = W_6$ .  
 If  $\sigma(W_6) = W_6$ , then  $p(B \cup \sigma(W_1) \cup \sigma(W_6)) = 1$ , and  $p(B \cup \sigma(S) \cup \sigma(W_5)) = p(B \cup \sigma(S) \cup \sigma(W_6)) = 0$ , for some  $\sigma(S) = W_1$ .  
 If  $\sigma(W_6) = W_1$ , then  $p(B \cup \sigma(W_2) \cup \sigma(W_6)) = 1$  and  $p(B \cup \sigma(W_6) \cup \sigma(W_5)) = 0$ .  
 Thus we always have  $\text{dist}(p, \sigma(p)) \geq 3$ .
- c.  $\sigma(W_5) \neq W_5$  and  $\sigma(W_2) = W_2$ .  
 Note that  $p(B \cup \sigma(W_4) \cup \sigma(W_5)) = 0$  and  $p(B \cup \sigma(W_4) \cup \sigma(S)) = 1$  for some  $\sigma(S) = W_5$ .  
 If  $\sigma(W_5) \in D \cup \{W_1\}$ , then  $p(B \cup \sigma(W_2) \cup \sigma(W_5)) = 1$ .  
 If  $\sigma(W_5) = W_6$  and  $\sigma(W_6) \in D \cup \{W_1\}$ , then  $p(B \cup \sigma(W_2) \cup \sigma(W_6)) = 1$ .  
 If  $\sigma(W_5) = W_6$  and  $\sigma(W_6) \in W_5$ , since  $\sigma(C) \neq C$ , thus  $\sigma(W_1) \in D$  and  $p(B \cup \sigma(W_1) \cup \sigma(W_5)) = 1$ .  
 Therefore we always have  $\text{dist}(p, \sigma(p)) \geq 3$ .
- d.  $\sigma(W_2) \neq W_2$  and  $\sigma(W_5) = W_5$ .  
 Similar to the above one.
4.  $\sigma(W_3) = W_4$  and  $\sigma(W_4) = W_3$ .  
 Similar to the case where  $\sigma(W_3) = W_3$  and  $\sigma(W_4) = W_4$ .

**Case for  $k_2 \geq 2$ .** Similarly, since  $|\{\sigma(p) | \sigma(C) = C \text{ and } \sigma(B) = B\}| = O(1)$ , we only need to consider the set  $\{\sigma(p) | \sigma(C) \neq C \text{ and } \sigma(B) = B\}$ , and we exclude each  $\sigma$  case by case:

1.  $\sigma(W_3)$  or  $\sigma(W_4) \notin \{W_3, W_4\}$ .  
 Assume  $\sigma(W_3) \notin \{W_3, W_4\}$ , note that for any  $S \cap (B \cup W_3) = \emptyset$ ,  $p(B \cup W_3 \cup S) \neq \star$  and there are only two such  $S$ s to make  $p = 1$ . While no matter what  $\sigma(W_3)$  is, it's easy to see there are at least five (actually many) such  $S$ s to make  $p(B \cup \sigma(W_3) \cup \sigma(S)) = 1$ , thus  $\text{dist}(p, \sigma(p)) \geq 3$ .
2.  $\sigma(W_3), \sigma(W_4) \in \{W_3, W_4\}$   
 W.l.o.g, assume  $\sigma(W_3) = W_3$  and  $\sigma(W_4) = W_4$ .
  - a.  $\sigma(W_5) \neq W_5$  and  $\sigma(W_2) \neq W_2$ .  
 Now  $p(B \cup \sigma(W_3) \cup \sigma(W_2)) = p(B \cup \sigma(W_4) \cup \sigma(W_5)) = 0$  and  $p(B \cup \sigma(W_3) \cup \sigma(S)) = p(B \cup \sigma(W_4) \cup \sigma(S')) = 1$ , for some  $\sigma(S) = W_2$  and  $\sigma(S') = W_5$ . Therefore,  $\text{dist}(p, \sigma(p)) \geq 4$ .
  - b.  $\sigma(W_5) = W_5$  or  $\sigma(W_2) = W_2$ .

Assume  $\sigma(W_5) = W_5$ , since  $\sigma(C) \cap D \neq \emptyset$ , there exists some  $W \in \{W_1, W_2, W_6\}$  such that  $\sigma(W) \cap D \neq \emptyset$ .

Moreover, for any  $S \subseteq W_3 \cup W_4 \cup W_5$  and  $S \notin \{W_3, W_4, W_5\}$  with  $|S| = k_2$ , we have  $\sigma(S) \subseteq W_3 \cup W_4 \cup W_5$  and  $\sigma(S) \notin \{W_3, W_4, W_5\}$ , thus  $p(B \cup W \cup S) = 0 \neq p(B \cup \sigma(W) \cup \sigma(S)) = 1$ , and note that there are at least  $\binom{6}{2} - 3 = 12$  such  $S$ s. Thus,  $\text{dist}(p, \sigma(p)) \geq 3$ . ◀

► **Theorem 2 (restated).** *For any fixed  $k \geq 4$  satisfying  $k \equiv 1 \pmod{3}$ , there exists a sequence of  $k$ -uniform hypergraph properties  $f$  such that  $s(f) = O(n^{\lceil k/3 \rceil - 1/2})$ , where  $n$  is the number of vertices.*

**Proof.** We still use minterm functions here.

Let  $k = 3l + 1$ . Note that in the above construction for  $(3l + 1)$ -uniform hypergraph properties,  $s_1(f) \leq |p| = O(n^l)$  and  $s_0(f) = O(n^{l+1})$ . Intuitively, we can pack  $\sqrt{n}$  minterms together to get a super minterm, expecting to decrease the number of shifts of  $p$  satisfying the distance constraint (i.e., where any of two shifts of  $p$  have distance at most 2). However, just packing minterms naively doesn't work here, we need to do more.

Let  $p$  be the minterm defining  $f$ .  $p$  is constructed as follows:

The notions  $V, B, W_i, C$  and  $D$  are defined the same as in Theorem 1, where we let  $k_1 = k_2 = l$ . Besides that, let  $D_1 = \{v_{6l+1}, v_{6l+2}, \dots, v_{6l+\sqrt{n}}\}$  and  $D_2 = D \setminus D_1$ .

- For any  $S \subseteq C$  of size  $2l$  and any  $v \in D_1$ ,  $p(B \cup S \cup v) = 0$ , except  $S = W_i \cup W_{i+1}$  for  $i \in [5]$  where  $p(B \cup S \cup v) = 1$ .
- For any  $S \subseteq C$  of size  $2l$  and any  $v \in D_2$ ,  $p(B \cup S \cup v) = 1$ .
- For any  $S$  satisfying  $l \leq |S \cap C| < 2l$ ,  $|S| = 2l + 1$  and  $S \cap D_1 \neq \emptyset$ ,  $p(B \cup S) = 1$ , except  $W_3$  or  $W_4 \subseteq S$  where  $p(B \cup S) = 0$ .
- All the other variables are  $\star$ .

It is not hard to see that  $|p| = O(n^{l+1/2})$ , thus  $s_1(f) \leq C_1(f) \leq |p| = O(n^{l+1/2})$ .

Similar to the argument in the proof of Theorem 1, we just need to show the following claim to complete the proof.

► **Claim 19.** *There are only  $O(\sqrt{n})$   $\sigma(p)$ s with the same  $\pi(B) = \sigma(B)$  satisfying  $\text{dist}(\pi(p), \sigma(p)) \leq 2$ .*

**Proof.** By contradiction, suppose there are  $C\sqrt{n}$  such  $\sigma(p)$ s where  $C$  is a sufficient large number, thus there must exist a vertex  $v$  such that  $\sigma(v) \in D_1$  for at least  $C$  such  $\sigma(p)$ s, w.l.o.g, assume this set contains  $p$ . And we will argue that there are only  $O(1)$  such  $\sigma(p)$ s satisfying  $\text{dist}(\sigma(p), p) \leq 2$ , then it's a contradiction, which completes the proof. ◀

Since the vertices in  $D_1$  or  $D_2$  are symmetric, thus  $|\{\sigma(p) | \sigma(C) = C \text{ and } \sigma(D_1) = D_1\}| = O(1)$ .

If  $\sigma(C) = C$  and  $\exists v_1 \in D_1, v_2 \in D_2$  satisfying  $\sigma(v_1) = v_2$ , then  $\text{dist}(\sigma(p), p) \geq 3$ , since almost all variables which contains  $v_1, C$  and  $B$  are 0 in  $p$ , while all these variable are 1 in  $\sigma(p)$ .

If  $\sigma(C) \neq C$ , since  $\sigma(v) \in D_1$ , then we find that  $p(S \cup v) = p'(S)$  where  $p'$  is the minterm defined in Theorem 1 for  $3l$ -uniform hypergraph properties. Similarly,  $\sigma(p)(S \cup v) = p(\sigma(S) \cup \sigma(v)) = p'(\sigma(S))$ . We only consider those  $S$ s satisfying  $v \notin \sigma(S) \cup S$  and follows the similar proof of Claim 1 in Theorem 1. Finally we can obtain  $\text{dist}(p, \sigma(p)) \geq 3$ . ◀

■ **Table 1** The table to illustrate  $p$  of  $k$ -partite  $k$ -uniform hypergraph properties.

$\vec{b} =$	$\vec{z}_1$	$\vec{z}_2$	$\vec{z}_3$	$\vec{z}_4$	$\vec{z}_5$	$\vec{z}_6$	$\vec{z}_7$	$\dots$
$\vec{a} = \vec{z}_1$	0	0	0	1	1	1	1	1
$\vec{z}_2$	0	0	1	1	1	1	1	1
$\vec{z}_3$	1	0	1	0	0	0	0	0
$\vec{z}_4$	1	1	0	*	*	*	*	*
$\vec{z}_5$	1	1	0	*	*	*	*	*
$\vec{z}_6$	1	1	0	*	*	*	*	*
$\vec{z}_7$	1	1	0	*	*	*	*	*
$\dots$	1	1	0	*	*	*	*	*

#### 4 $k$ -Partite $k$ -Uniform Hypergraph Properties and Abelian Groups

In this section, we give the constructions of  $k$ -partite  $k$ -uniform hypergraph properties first.

► **Theorem 3 (restated).** *For any  $k \geq 3$ , there exists a sequence of  $k$ -partite  $k$ -uniform hypergraph properties  $f : \{0, 1\}^{n^k} \rightarrow \{0, 1\}$  such that  $s(f) = O(n^{\lceil k/3 \rceil})$ .*

**Proof.** The function we use here is also a minterm function. Let  $k_1$  and  $k_2$  be the integers such that  $k_1 + 2k_2 = k$  and  $\lceil k/3 \rceil \leq k_1, k_2 \leq \lceil k/3 \rceil$ . We divide the  $k$  partitions into three sets, and each of them is of size  $k_2$ ,  $k_2$  and  $k_1$ . And they are indicated by  $\vec{a}, \vec{b} \in [n]^{k_2}$  and  $\vec{c} \in [n]^{k_1}$  respectively. Assume  $t$  is an integer, let  $\vec{z}_t$  ( $\vec{z}_t^c$  respectively) be the  $t$ -th smallest vector in  $[n]^{k_2}$  ( $[n]^{k_1}$  respectively) in the lexicographic order. We use Table 1 to illustrate the minterm  $p$ :

- For  $\vec{b} = \vec{z}_1, \vec{z}_2$  or  $\vec{z}_3$ ,  $p(\vec{z}_1, \vec{b}, \vec{z}_1^c) = 0$ , otherwise  $p(\vec{z}_1, \vec{b}, \vec{z}_1^c) = 1$ .
- For  $\vec{b} = \vec{z}_1$  or  $\vec{z}_2$ ,  $p(\vec{z}_2, \vec{b}, \vec{z}_1^c) = 0$ , otherwise  $p(\vec{z}_2, \vec{b}, \vec{z}_1^c) = 1$ .
- For  $\vec{b} = \vec{z}_1$  or  $\vec{z}_3$ ,  $p(\vec{z}_3, \vec{b}, \vec{z}_1^c) = 1$ , otherwise  $p(\vec{z}_3, \vec{b}, \vec{z}_1^c) = 0$ .
- For  $\vec{a} \notin \{\vec{z}_1, \vec{z}_2, \vec{z}_3\}$  and  $\vec{b} = \vec{z}_1$  or  $\vec{z}_2$ ,  $p(\vec{a}, \vec{b}, \vec{z}_1^c) = 1$ .
- For  $\vec{a} \notin \{\vec{z}_1, \vec{z}_2, \vec{z}_3\}$  and  $\vec{b} = \vec{z}_3$ ,  $p(\vec{a}, \vec{b}, \vec{z}_1^c) = 0$ .
- Otherwise  $p(\vec{a}, \vec{b}, \vec{c}) = *$ .

It's easy to see  $s_1(f) \leq C_1(f) \leq |p| = O(n^{k_2})$ . By discussing case by case, it can be verified that for any  $p^\pi$  there are at most  $O(1)$   $p^\sigma$ s satisfying  $\pi(\vec{c}) = \sigma(\vec{c})$  and  $\text{dist}(p^\pi, p^\sigma) \leq 2$ . Observe that there are  $n^{k_1}$  choices of  $\pi(\vec{c})$ , thus there are at most  $O(n^{k_1})$   $p^\sigma$ s such that the distance between any two of them is at most 2. Similar to the argument in the proof of Theorem 1, we can conclude  $s_0(f) = O(n^{k_1})$ . The verifying procedure is straightforward but tedious, and we omit it here. ◀

In the following, we give the proofs of Theorem 4 and Theorem 5.

► **Theorem 4 (restated).** *For any  $k \geq 4$  satisfying  $k \equiv 1 \pmod{3}$ , there exists a sequence of  $k$ -partite  $k$ -uniform hypergraph properties  $f : \{0, 1\}^{n^k} \rightarrow \{0, 1\}$  such that  $s(f) = O(n^{\lceil k/3 \rceil - 1/2})$ .*

**Proof.** We still use minterm functions here. Let  $k = 3l + 1$  where  $l \geq 1$ . We divide the  $k$  partitions into four sets of size  $l$ ,  $l$ ,  $l$  and 1, and each set is indicated by  $\vec{a}, \vec{b}, \vec{c} \in [n]^l$  and  $\vec{d} \in [n]$  respectively. Assume  $t$  is an integer, let  $\vec{z}_t$  be the  $t$ -th smallest vector in  $[n]^l$  in the lexicographic order. The minterm  $p$  is constructed as follows:

- For any  $\vec{d} \in [\lceil \sqrt{n} \rceil]$ , and any  $\vec{a}$  and  $\vec{b}$ ,  $p(\vec{a}, \vec{b}, \vec{z}_1, \vec{d}) = p'(\vec{a}, \vec{b}, \vec{z}_1)$ . Here  $p'$  is the partial assignment defined in the proof of Theorem 3.

- For any  $\vec{d} \notin [\lceil \sqrt{n} \rceil]$  and  $\vec{a}, \vec{b} \in \{\vec{z}_1, \vec{z}_2, \vec{z}_3\}$ ,  $p(\vec{a}, \vec{b}, \vec{z}_1, \vec{d}) = 1$ .
- Otherwise  $p(\vec{a}, \vec{b}, \vec{c}, \vec{d}) = \star$ .

It's easy to see  $s_1(f) \leq |p| = O(n^{l+1/2})$ . It is also not hard to verify that there are at most  $\lceil \sqrt{n} \rceil$   $p^\sigma$ s with the same  $\sigma(\vec{c})$  and satisfying the condition that the distance between any two of them is at most 2, thus  $s_0(f) = O(n^{l+1/2})$ . ◀

► **Theorem 5 (restated).** *Let  $G \leq S_n$  be a transitive Abelian group, then there exists a Boolean function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  invariant under  $G$  such that  $s(f) \leq \alpha n^{1/3}$ , where  $\alpha$  is a number only depending on  $l$ .*

**Proof.** First note that the transitive action of a group  $G$  on  $[n]$  is equivalent to the action of  $G$  by left multiplication on a coset space  $G/\text{Stab}_1$ , here  $\text{Stab}_1$  is the stabilizer of the element  $1 \in [n]$ . Since  $G$  is an Abelian group,  $\text{Stab}_1 = \dots = \text{Stab}_n$ , thus  $\text{Stab}_1 = \{e\}$ . Therefore, the action of  $G$  on  $[n]$  is equivalent to the action of  $G$  by multiplication on itself. So we can relabel the variables  $(x_1, \dots, x_n)$  as  $(x_{(1, \dots, 1)}, \dots, x_{(m_1, \dots, m_l)})$  to make  $(\sigma_1 \otimes \dots \otimes \sigma_l)(x) = (x_{(\sigma_1(1), \dots, \sigma_l(1))}, \dots, x_{(\sigma(m_1), \dots, \sigma(m_l))})$  for any  $\sigma_1 \otimes \dots \otimes \sigma_l \in C_{m_1} \times \dots \times C_{m_l}$ .

Let  $p_m$  be the minterm of  $f : \{0, 1\}^m \rightarrow \{0, 1\}$  defined by Chakraborty in Theorem 3.1 in [14]. We define the minterm  $p$  as  $p(i_1, \dots, i_l) = \bigoplus_{j=1}^l p_{m_j}(i_j)$ . Here  $\star \oplus b = \star$ , for  $b = 0, 1$ , or  $\star$ . It is easy to see  $s_1(f) \leq |p| = \prod_{j=1}^l |p_{m_j}| \leq \gamma n^{1/3}$ , where  $\gamma$  is a number only depending on  $l$ . Moreover, according to the construction of  $p_m$ , it is easy to see that there are at most  $\beta n^{1/3}$   $\sigma(p)$ s where the distance between any two of them is at most 2. Here  $\beta$  is another number only depending on  $l$ , thus  $s_0(f) \leq \beta n^{1/3}$ . This completes the proof. ◀

## 5 Lower bounds

In this section, we give the proof of Theorem 6.

► **Theorem 6 (restated).** *For any fixed  $k$  and any non-trivial  $k$ -uniform hypergraph property  $f$ ,  $s(f) = \Omega(n)$ , where  $n$  is the number of vertices.*

**Proof.** W.l.o.g we assume that for the empty graph  $\overline{K}_n$ ,  $f(\overline{K}_n) = 0$ . Since  $f$  is non-trivial, there must exist a graph  $G$  such that  $f(G) = 1$ . Let's consider graphs in  $f^{-1}(1) = \{G | f(G) = 1\}$  with the minimum number of edges. Define  $m = \min\{|E(G)| : f(G) = 1\}$ .

We claim that if  $m \geq \frac{1}{k+2}n$ , then  $s(f) \geq \frac{1}{k+2}n$ . Let  $G$  be a graph in  $f^{-1}(1)$  and  $|E(G)| = m$ . Consider the subfunction  $f'$  where  $\forall e \notin E(G)$ ,  $x_e$  is restricted to 0, since  $G$  has the the minimum number of edges, deleting any edges from  $G$  will change the values of  $f(G)$ , therefore,  $f'$  is an AND function. Thus,  $s(f) \geq s(f') = m \geq \frac{1}{k+2}n$ .

In the following we assume  $m < \frac{1}{k+2}n$ . Again let  $G$  be a graph in  $f^{-1}(1)$  with  $|E(G)| = m$ . Let us consider the isolated vertices set  $I$ , as  $\sum_{v \in V} \deg(v) = k|E(G)| < \frac{k}{k+2}n$ , we have  $|I| \geq n - \sum_{v \in V} \deg(v) > \frac{2}{k+2}n$ . Suppose  $s(f) < \frac{1}{k+2}n$ , we will deduce that there exists another graph with fewer edges and the same value, against the assumption that  $G$  has the minimum number of edges in  $f^{-1}(1)$ , which ends the whole proof.

Pick a vertex  $u$  with  $\deg(u) = d > 0$ . Suppose in the graph  $G$  vertex  $u$  is adjacent to  $(k-1)$ -edges  $\{e_1^{(k-1)}, e_2^{(k-1)}, \dots, e_d^{(k-1)}\}$  and  $I = \{u_1, u_2, \dots, u_t\}$ , where  $t = |I|$ .

Consider the  $t$ -variable Boolean function  $g_1 : \{0, 1\}^t \rightarrow \{0, 1\}$ , where

$$g_1(x_1, \dots, x_t) = f(G + x_1(e_1^{(k-1)}, u_1) + \dots + x_t(e_1^{(k-1)}, u_t)).$$

It is easy to see that  $g_1$  is a symmetric function. We claim that  $g_1$  is a constant function: if not, we have  $s(g_1) \geq \frac{1}{2}t$  [38], which implies  $s(f) > \frac{1}{k+2}n$  since  $g_1$  is a restriction of  $f$ . In particular,  $g_1(1, \dots, 1) = g_1(0, \dots, 0)$ , i.e.  $f(G_1) = f(G)$ , where  $G_1 = G + \sum_{i=1}^t (e_1^{(k-1)}, u_i)$ .

## 51:10 On the Sensitivity Complexity of $k$ -Uniform Hypergraph Properties

Define  $G_i = G_{i-1} + \sum_{j=1}^t (e_i^{(k-1)}, u_j)$  ( $i = 2, \dots, d$ ). Similarly, we can show that

$$f(G) = f(G_1) = \dots = f(G_d).$$

Next we will delete all the edges between  $\{u, u_1, \dots, u_t\}$  and  $\{e_1^{(k-1)}, e_2^{(k-1)}, \dots, e_d^{(k-1)}\}$  from  $G_d$  by reversing the adding edge procedure of  $G \rightarrow G_1 \rightarrow \dots \rightarrow G_d$ . More precisely, define  $H_1 = G_d$ ; for  $i = 2, \dots, d$ , define

$$H_i = H_{i-1} - (e_i^{(k-1)}, u) - (e_i^{(k-1)}, u_1) - \dots - (e_i^{(k-1)}, u_t),$$

and

$$h_i(y_0, y_1, \dots, y_t) = f(H_i + y_0(e_i^{(k-1)}, u) + y_1(e_i^{(k-1)}, u_1) + \dots + y_t(e_i^{(k-1)}, u_t)).$$

Similarly, by the fact  $s(f) < \frac{1}{k+2}n$  we can show that all the functions  $h_2, \dots, h_d$  are constant, which implies  $f(H_1) = f(H_2) = \dots = f(H_d)$ . So we find another graph  $H_d$  with fewer edges than  $G$  and  $f(H_d) = 1$ . ◀

## 6 Conclusion

In this paper, we present a  $k$ -uniform hypergraph property with sensitivity complexity  $O(n^{\lceil k/3 \rceil})$  for any  $k \geq 3$  and we can do better when  $k \equiv 1 \pmod{3}$ . Besides that, we also investigate the sensitivity complexity of other transitive Boolean functions with certain symmetry. All the functions we constructed in this paper are minterm transitive functions. On the other side, Chakraborty [14] proved that the sensitivity complexity of any minterm transitive Boolean function  $f : \{0, 1\}^N \rightarrow \{0, 1\}$  is at least  $\Omega(N^{1/3})$ . Kulkarni et al. [27] pointed out that the existence of any transitive function  $f : \{0, 1\}^N \rightarrow \{0, 1\}$  with  $s(f) = N^\alpha$  where  $\alpha < 1/3$  implies a larger than quadratic separation between block sensitivity and sensitivity. We conjecture that the examples here are almost tight.

► **Conjecture 20.** For any  $k \geq 3$  and for any non-trivial  $k$ -hypergraph property  $f$ ,  $s(f) = \Omega(n^{k/3})$ , where  $n$  is the number of vertices.

► **Conjecture 21.** For any  $k \geq 3$ , there exists a sequence of  $k$ -uniform hypergraph properties  $f$  with  $s(f) = O(n^{k/3})$ , where  $n$  is the number of vertices.

A more general question is the following variant of Turan's question proposed by Chakraborty [14]: If  $f : \{0, 1\}^N \rightarrow \{0, 1\}$  is Boolean function invariant under a transitive group of permutations, then is it true that  $s(f) = \Omega(N^c)$  for some constant  $c > 0$ ? We conjecture that the inequality holds for  $c = 1/3$ , which would imply Conjecture 20 and the sensitivity conjecture of transitive functions.

---

## References

- 1 Andris Ambainis, Mohammad Bavarian, Yihan Gao, Jieming Mao, Xiaoming Sun, and Song Zuo. Tighter relations between sensitivity and other complexity measures. In *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 101–113, 2014.
- 2 Andris Ambainis and Krisjanis Prusis. A tight lower bound on certificate complexity in terms of block sensitivity and sensitivity. In *Mathematical Foundations of Computer Science 2014 – 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, pages 33–44, 2014.

- 3 Andris Ambainis, Krisjanis Prusis, and Jevgenijs Vihrovs. Sensitivity versus certificate complexity of boolean functions. In *Computer Science – Theory and Applications – 11th International Computer Science Symposium in Russia, CSR 2016, St. Petersburg, Russia, June 9–13, 2016, Proceedings*, pages 16–28, 2016.
- 4 Andris Ambainis and Xiaoming Sun. New separation between  $s(f)$  and  $bs(f)$ . *Electronic Colloquium on Computational Complexity (ECCC)*, 18:116, 2011.
- 5 Andris Ambainis and Jevgenijs Vihrovs. Size of sets with small sensitivity: A generalization of simon’s lemma. In *Theory and Applications of Models of Computation – 12th Annual Conference, TAMC 2015, Singapore, May 18–20, 2015, Proceedings*, pages 122–133, 2015.
- 6 László Babai, Anandam Banerjee, Raghav Kulkarni, and Vipul Naik. Evasiveness and the distribution of prime numbers. In *27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, March 4–6, 2010, Nancy, France*, pages 71–82, 2010.
- 7 Mitali Bafna, Satyanarayana V. Lokam, Sébastien Tavenas, and Ameya Velingker. On the sensitivity conjecture for read- $k$  formulas. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22–26, 2016 – Kraków, Poland*, pages 16:1–16:14, 2016.
- 8 Shalev Ben-David. Low-sensitivity functions from unambiguous certificates. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:84, 2016.
- 9 Stella Biderman, Kevin Cuddy, Ang Li, and Min Jae Song. On the sensitivity of  $k$ -uniform hypergraph properties. *CoRR*, abs/1510.00354, 2015.
- 10 Timothy Black. Monotone properties of  $k$ -uniform hypergraphs are weakly evasive. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS’15*, pages 383–391, 2015.
- 11 Meena Boppana. Lattice variant of the sensitivity conjecture. *Electronic Colloquium on Computational Complexity (ECCC)*, 19:89, 2012.
- 12 Harry Buhrman and Ronald De Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21–43, 2002.
- 13 Amit Chakrabarti, Subhash Khot, and Yaoyun Shi. Evasiveness of subgraph containment and related properties. *SIAM J. Comput.*, 31(3):866–875, 2001.
- 14 Sourav Chakraborty. On the sensitivity of cyclically-invariant boolean functions. In *Proceedings of the 20th Annual IEEE Conference on Computational Complexity, CCC’05*, pages 163–167, 2005.
- 15 Sourav Chakraborty. Sensitivity, Block Sensitivity and Certificate Complexity of Boolean Functions. Master’s thesis, University of Chicago, 2005.
- 16 Stephen Cook and Cynthia Dwork. Bounds on the time for parallel ram’s to compute simple functions. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, STOC’82*, pages 231–233, 1982.
- 17 Stephen A. Cook, Cynthia Dwork, and Rüdiger Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM J. Comput.*, 15(1):87–97, 1986.
- 18 Yihan Gao, Jieming Mao, Xiaoming Sun, and Song Zuo. On the sensitivity complexity of bipartite graph properties. *Theor. Comput. Sci.*, 468:83–91, 2013.
- 19 Justin Gilmer, Michal Koucký, and Michael E. Saks. A new approach to the sensitivity conjecture. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11–13, 2015*, pages 247–254, 2015.
- 20 Parikshit Gopalan, Noam Nisan, Rocco A. Servedio, Kunal Talwar, and Avi Wigderson. Smooth boolean functions are easy: Efficient algorithms for low-sensitivity functions. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14–16, 2016*, pages 59–70, 2016.

- 21 Parikshit Gopalan, Rocco A. Servedio, and Avi Wigderson. Degree and sensitivity: Tails of two distributions. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 13:1–13:23, 2016.
- 22 Pooya Hatami, Raghav Kulkarni, and Denis Pankratov. Variations on the sensitivity conjecture. *Theory of Computing, Graduate Surveys*, 4:1–27, 2011.
- 23 Kun He, Qian Li, and Xiaoming Sun. A tighter relation between sensitivity and certificate complexity. *CoRR*, abs/1609.04342, 2016.
- 24 Ilan Karpas. Lower bounds for sensitivity of graph properties. *CoRR*, abs/1609.05320, 2016.
- 25 Raghav Kulkarni. Evasiveness through a circuit lens. In *Innovations in Theoretical Computer Science, ITCS'13, Berkeley, CA, USA, January 9-12, 2013*, pages 139–144, 2013.
- 26 Raghav Kulkarni, Youming Qiao, and Xiaoming Sun. Any monotone property of 3-uniform hypergraphs is weakly evasive. *Theor. Comput. Sci.*, 588:16–23, 2015.
- 27 Raghav Kulkarni and Avishay Tal. On fractional block sensitivity. *Chicago J. Theor. Comput. Sci.*, 2016, 2016.
- 28 Chengyu Lin and Shengyu Zhang. Sensitivity conjecture and log-rank conjecture for functions with small alternating numbers. *CoRR*, abs/1602.06627, 2016.
- 29 László Lovász and Neal E. Young. Lecture notes on evasiveness of graph properties. *CoRR*, cs.CC/0205031, 2002.
- 30 Noam Nisan. Crew prams and decision trees. *SIAM Journal on Computing*, 20(6):999–1007, 1991.
- 31 Noam Nisan and Mario Szegedy. On the degree of boolean functions as real polynomials. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing, STOC'92*, pages 462–467, 1992.
- 32 Ronald L. Rivest and Jean Vuillemin. On recognizing graph properties from adjacency matrices. *Theor. Comput. Sci.*, 3(3):371–384, 1976.
- 33 Karthik C. S. and Sébastien Tavenas. On the sensitivity conjecture for disjunctive normal forms. *CoRR*, abs/1607.05189, 2016.
- 34 Xiaoming Sun. Block sensitivity of weakly symmetric functions. *Theor. Comput. Sci.*, 384(1):87–91, 2007.
- 35 Xiaoming Sun. An improved lower bound on the sensitivity complexity of graph properties. *Theoretical Computer Science*, 412(29):3524–3529, 2011.
- 36 Mario Szegedy. An  $o(n^{0.4732})$  upper bound on the complexity of the GKS communication game. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:102, 2015.
- 37 Avishay Tal. On the sensitivity conjecture. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 38:1–38:13, 2016.
- 38 György Turán. The critical complexity of graph properties. *Information Processing Letters*, 18(3):151–153, 1984.



# The Complexity of Knapsack in Graph Groups

Markus Lohrey<sup>1</sup> and Georg Zetsche<sup>\*2</sup>

1 Universität Siegen, Germany  
lohrey@eti.uni-siegen.de

2 LSV, CNRS & ENS Cachan, Université Paris-Saclay, France  
zetsche@lsv.fr

---

## Abstract

Myasnikov et al. have introduced the knapsack problem for arbitrary finitely generated groups. In [19] the authors proved that for each graph group, the knapsack problem can be solved in NP. Here, we determine the exact complexity of the problem for every graph group. While the problem is  $TC^0$ -complete for complete graphs, it is LogCFL-complete for each (non-complete) transitive forest. For every remaining graph, the problem is NP-complete.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** knapsack, subset sum, graph groups, decision problems in group theory

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.52

## 1 Introduction

In their paper [21], Myasnikov, Nikolaev, and Ushakov started the investigation of discrete optimization problems, which are classically formulated over the integers, for arbitrary (possibly non-commutative) groups. The general goal of this line of research is to study to what extent results from the classical commutative setting can be transferred to the non-commutative setting. Among other problems, Myasnikov et al. introduced for a finitely generated group  $G$  the *knapsack problem* and the *subset sum problem*. The input for the knapsack problem is a sequence of group elements  $g_1, \dots, g_k, g \in G$  (specified by finite words over the generators of  $G$ ) and it is asked whether there exists a solution  $(x_1, \dots, x_k) \in \mathbb{N}^k$  of the equation  $g_1^{x_1} \cdots g_k^{x_k} = g$ . For the subset sum problem one restricts the solution to  $\{0, 1\}^k$ . For the particular case  $G = \mathbb{Z}$  (where the additive notation  $x_1 \cdot g_1 + \cdots + x_k \cdot g_k = g$  is usually preferred) these problems are NP-complete if the numbers  $g_1, \dots, g_k, g$  are encoded in binary representation. For subset sum, this is a classical result from Karp's seminal paper [15] on NP-completeness. Knapsack for integers is usually formulated in a more general form in the literature; NP-completeness of the above form (for binary encoded integers) was shown in [11], where the problem was called MULTISUBSET SUM.<sup>1</sup> Interestingly, if we consider subset sum for the group  $G = \mathbb{Z}$ , but encode the input numbers  $g_1, \dots, g_k, g$  in unary notation, then the problem is in DLOGTIME-uniform  $TC^0$  (a small subclass of polynomial time and even of logarithmic space that captures the complexity of multiplication of binary encoded numbers; see e.g. the book [25] for more details) [7], and the same holds for knapsack (see Theorem 1). Related results can be found in [13]. Implicitly, the knapsack problem was also

---

\* This author is supported by a fellowship within the Postdoc-Program of the German Academic Exchange Service (DAAD).

<sup>1</sup> Note that if we ask for a solution  $(x_1, \dots, x_k)$  in  $\mathbb{Z}^k$ , then knapsack can be solved in polynomial time (even for binary encoded integers) by checking whether  $\gcd(g_1, \dots, g_k)$  divides  $g$ .



studied by Babai et al. [3], where it is shown that knapsack for commutative matrix groups over algebraic number fields can be solved in polynomial time.

In [21] the authors encode elements of the finitely generated group  $G$  by words over the group generators and their inverses, which corresponds to the unary encoding of integers. Another, more succinct encoding of group elements uses *straight-line programs* (SLP). These are context-free grammars that produce a single word. Over a unary alphabet, one can achieve for every word exponential compression with SLPs: The word  $a^n$  can be produced by an SLP of size  $O(\log n)$ . This shows that knapsack and subset sum for the group  $\mathbb{Z}$  with SLP-compressed group elements correspond to the classical knapsack and subset sum problem with binary encoded numbers. To distinguish between the two variants, we will speak in this introduction of uncompressed knapsack (resp., subset sum) if the input group elements are given explicitly by words over the generators. On the other hand, if these words are represented by SLPs, we will speak of SLP-compressed knapsack (resp., subset sum). Later in this paper, we will only use the uncompressed versions, and denote these simply with knapsack and subset sum, respectively.

In our recent paper [19], we started to investigate knapsack and subset sum for graph groups, which are also known as right-angled Artin groups in group theory. A graph group is specified by a finite simple graph  $\Gamma$  and denoted with  $\mathbb{G}(\Gamma)$ . The vertices are the generators of the group, and two generators  $a$  and  $b$  are allowed to commute if and only if  $a$  and  $b$  are adjacent in  $\Gamma$ . Graph groups interpolate between free groups and free abelian groups and can be seen as a group counterpart of trace monoids (free partially commutative monoids), which have been used for the specification of concurrent behavior. In combinatorial group theory, graph groups are currently an active area of research, mainly because of their rich subgroup structure, see e.g. [4, 5, 9].

**Contribution.** In [19] we proved that for every graph group, SLP-compressed knapsack (resp., subset sum) is NP-complete. This result generalizes the classical result for knapsack with binary encoded integers. Moreover, we proved that uncompressed knapsack and subset sum are NP-complete for the group  $F_2 \times F_2$  ( $F_2$  is the free group on two generators). The group  $F_2 \times F_2$  is the graph group  $\mathbb{G}(\Gamma)$ , where the graph  $\Gamma$  is a cycle on four nodes. This result leaves open the complexity of uncompressed knapsack and subset sum for graph groups, whose underlying graph does not contain an induced cycle on four nodes. In this paper, we completely settle this open problem for knapsack by showing the following results:

- (i) Uncompressed knapsack and subset sum for  $\mathbb{G}(\Gamma)$  are complete for  $\text{TC}^0$  if  $\Gamma$  is a complete graph (and thus  $\mathbb{G}(\Gamma)$  is a free abelian group).<sup>2</sup>
- (ii) Uncompressed knapsack and subset sum for  $\mathbb{G}(\Gamma)$  are LogCFL-complete if  $\Gamma$  is not a complete graph and neither contains an induced cycle on four nodes (C4) nor an induced path on four nodes (P4).
- (iii) Uncompressed knapsack for  $\mathbb{G}(\Gamma)$  is NP-complete if  $\Gamma$  contains an induced C4 or an induced P4.

**Overview of the proofs.** The result 1 is a straightforward extension of the corresponding result for  $\mathbb{Z}$  [7]. The statements in 2 and 3 are less obvious. Recall that LogCFL is the closure of the context-free languages under logspace reductions; it is contained in  $\text{NC}^2$ .

To show the upper bound in 2, we use the fact that the graph groups  $\mathbb{G}(\Gamma)$ , where  $\Gamma$  neither contains an induced C4 nor an induced P4 (these graphs are the so called transitive

---

<sup>2</sup> In the following,  $\text{TC}^0$  always refers to its DLOGTIME-uniform version.

forests), are exactly those groups that can be built up from  $\mathbb{Z}$  using the operations of free product and direct product with  $\mathbb{Z}$ . We then construct inductively over these operations a logspace-bounded auxiliary pushdown automaton working in polynomial time (these machines accept exactly the languages in **LogCFL**) that checks whether an acyclic finite automaton accepts a word that is trivial in the graph group. In order to apply this result to knapsack, we finally show that every solvable knapsack instance over a graph group  $\mathbb{G}(\Gamma)$  with  $\Gamma$  a transitive forest has a solution with polynomially bounded exponents. This result might be of independent interest.

For the lower bound in 2, it suffices to consider the group  $F_2$  (the free group on two generators). Our proof is based on the fact that the context-free languages are exactly those languages that can be accepted by valence automata over  $F_2$ . This is a reinterpretation of the classical theorem of Chomsky and Schützenberger. To the authors' knowledge, the result 2 is the first completeness result for **LogCFL** in the area of combinatorial group theory.

Finally, for the result 3 it suffices to show **NP**-hardness of knapsack for the graph group  $\mathbb{G}(\text{P4})$  (the **NP** upper bound and the lower bound for **C4** is shown in [19]). We apply a technique that was first used in a paper by Aalbersberg and Hooeboom [1] to show that the intersection non-emptiness problem for regular trace languages is undecidable for **P4**.

Full proofs can be found in the long version [18].

## 2 Knapsack and Exponent Equations

We assume that the reader has some basic knowledge concerning (finitely generated) groups (see e.g. [20] for further details). Let  $G$  be a finitely generated group, and let  $A$  be a finite generating set for  $G$ . Then, elements of  $G$  can be represented by finite words over the alphabet  $A^{\pm 1} = A \cup A^{-1}$ . An *exponent equation* over  $G$  is an equation of the form

$$h_0 g_1^{x_1} h_1 g_2^{x_2} h_2 \cdots g_k^{x_k} h_k = 1$$

where  $g_1, g_2, \dots, g_k, h_0, h_1, \dots, h_k \in G$  are group elements that are given by finite words over the alphabet  $A^{\pm 1}$  and  $x_1, x_2, \dots, x_k$  are not necessarily distinct variables. Such an exponent equation is *solvable* if there exists a mapping  $\sigma: \{x_1, \dots, x_k\} \rightarrow \mathbb{N}$  such that  $h_0 g_1^{\sigma(x_1)} h_1 g_1^{\sigma(x_2)} h_2 \cdots g_k^{\sigma(x_k)} h_k = 1$  in the group  $G$ . The *size* of an equation is  $\sum_{i=0}^k |h_i| + \sum_{i=1}^k |g_i|$ , where  $|g|$  denotes the length of the shortest word  $w \in (A^{\pm 1})^*$  representing  $g$ . *Solvability of exponent equations over  $G$*  is the following computational problem:

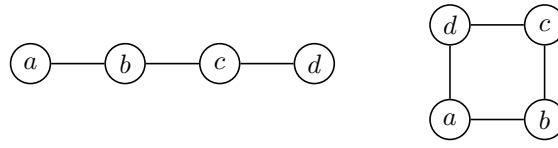
**Input:** An exponent equation  $E$  over  $G$  (with elements of  $G$  specified by words over  $A^{\pm 1}$ ).

**Question:** Is  $E$  solvable?

The *knapsack problem* for the group  $G$  is the restriction of solvability of exponent equations over  $G$  to exponent equations of the form  $g_1^{x_1} g_2^{x_2} \cdots g_k^{x_k} g^{-1} = 1$  or, equivalently,  $g_1^{x_1} g_2^{x_2} \cdots g_k^{x_k} = g$  where the exponent variables  $x_1, \dots, x_k$  have to be pairwise different. The *subset sum problem* for the group  $G$  is defined in the same way as the knapsack problem, but the exponent variables  $x_1, \dots, x_k$  have to take values in  $\{0, 1\}$ .

It is a simple observation that the decidability and complexity of solvability of exponent equations over  $G$  as well as the knapsack problem and subset sum problem for  $G$  does not depend on the chosen finite generating set for the group  $G$ . Therefore, we do not have to mention the generating set explicitly in these problems.

► **Remark.** Since we are dealing with a group, one might also allow solution mappings  $\sigma: \{x_1, \dots, x_k\} \rightarrow \mathbb{Z}$  to the integers. This variant of solvability of exponent equations (knapsack, respectively) can be reduced to the above version, where  $\sigma$  maps to  $\mathbb{N}$ , by simply replacing a power  $g_i^{x_i}$  by  $g_i^{x_i} (g_i^{-1})^{y_i}$ , where  $y_i$  is a fresh variable.



■ Figure 1 P4 and C4.

### 3 Traces and Graph Groups

Let  $(A, I)$  be a finite simple graph. In other words, the edge relation  $I \subseteq A \times A$  is irreflexive and symmetric. It is also called the *independence relation*, and  $(A, I)$  is called an *independence alphabet*. We say that  $a \in A$  *depends on*  $b \in A$  if  $(a, b) \notin I$ . We consider the monoid  $\mathbb{M}(A, I) = A^*/\equiv_I$ , where  $\equiv_I$  is the smallest congruence relation on the free monoid  $A^*$  that contains all pairs  $(ab, ba)$  with  $a, b \in A$  and  $(a, b) \in I$ . This monoid is called a *trace monoid* or *partially commutative free monoid*. Elements of  $\mathbb{M}(A, I)$  are called *Mazurkiewicz traces* or simply *traces*. The trace represented by the word  $u$  is denoted by  $[u]_I$ , or simply  $[u]$  if no confusion can arise. The empty trace  $[\varepsilon]_I$  is the identity element of the monoid  $\mathbb{M}(A, I)$  and is denoted by 1. For a language  $L \subseteq A^*$  we denote with  $[L]_I = \{[u]_I \in \mathbb{M}(A, I) \mid u \in L\}$  the set of traces represented by  $L$ . Figure 1 shows two important independence alphabets that we denote with P4 (path on four nodes) and C4 (cycle on four nodes). Note that  $\mathbb{M}(C4) = \{a, c\}^* \times \{b, d\}^*$ . For more details on traces see [6].

With an independence alphabet  $(A, I)$  we associate the finitely presented group  $\mathbb{G}(A, I) = \langle A \mid ab = ba \ ((a, b) \in I) \rangle$ . More explicitly, this group can be defined as follows: Let  $A^{-1} = \{a^{-1} \mid a \in A\}$  be a disjoint copy of the alphabet  $A$ . We extend the independence relation  $I$  to  $A^{\pm 1} = A \cup A^{-1}$  by  $(a^x, b^y) \in I$  for all  $(a, b) \in I$  and  $x, y \in \{-1, 1\}$ . Then  $\mathbb{G}(A, I)$  is the quotient monoid  $(A^{\pm 1})^*/\sim_I$ , where  $\sim_I$  is the smallest congruence relation that contains (i) all pairs  $(ab, ba)$  for  $a, b \in A^{\pm 1}$  with  $(a, b) \in I$  and (ii) all pairs  $(aa^{-1}, \varepsilon)$  and  $(a^{-1}a, \varepsilon)$  for  $a \in A$ .

A group  $\mathbb{G}(A, I)$  is called a *graph group*, or *right-angled Artin group*<sup>3</sup>, or *free partially commutative group*. Here, we use the term graph group. Graph groups received a lot of attention in group theory during the last years, mainly due to their rich subgroup structure [4, 5, 9], and their relationship to low dimensional topology [2, 12, 26].

### 4 TC<sup>0</sup>- and LogCFL-completeness

Let us first consider free abelian groups  $\mathbb{Z}^m$ . Note that  $\mathbb{Z}^m$  is isomorphic to the graph group  $\mathbb{G}(A, I)$  where  $(A, I)$  is the complete graph on  $m$  nodes. Our first result is a simple combination of known results [7, 22].

► **Theorem 1.** *For every fixed  $m \geq 1$ , knapsack and subset sum for the free abelian group  $\mathbb{Z}^m$  are complete for TC<sup>0</sup>. Hence, knapsack and subset sum for  $\mathbb{G}(A, I)$  are complete for TC<sup>0</sup> if  $(A, I)$  is a non-empty complete graph.*

We now characterize those graph groups where knapsack for  $\mathbb{G}(A, I)$  is LogCFL-complete. The class LogCFL consists of all problems that are logspace reducible to a context-free

<sup>3</sup> This term comes from the fact that right-angled Artin groups are exactly the Artin groups corresponding to right-angled Coxeter groups.

language. It is included in the parallel complexity class  $\text{NC}^2$  and has several alternative characterizations (see e.g. [24, 25]).

The *comparability graph* of a rooted tree  $t$  is the simple graph with the same vertices as  $t$ , but has an edge between two vertices whenever one is a descendent of the other in  $t$ . A graph  $(A, I)$  is a *transitive forest* if it is a disjoint union of comparability graphs of trees.

► **Theorem 2.** *If  $(A, I)$  is a transitive forest and not complete, then knapsack and subset sum for  $\mathbb{G}(A, I)$  are LogCFL-complete.*

If the graph  $(A, I)$  is the disjoint union of graphs  $\Gamma_0$  and  $\Gamma_1$ , then by definition, we have  $\mathbb{G}(A, I) \cong \mathbb{G}(\Gamma_0) * \mathbb{G}(\Gamma_1)$ . If one vertex  $v$  of  $(A, I)$  is adjacent to every other vertex and removing  $v$  from  $(A, I)$  results in the graph  $\Gamma_0$ , then  $\mathbb{G}(A, I) \cong \mathbb{G}(\Gamma_0) \times \mathbb{Z}$ . Therefore, we have the following *inductive characterization* of the graph groups  $\mathbb{G}(A, I)$  for transitive forests  $(A, I)$ : It is the smallest class of groups containing the trivial group that is closed under taking

- (i) free products and
- (ii) direct products with  $\mathbb{Z}$ .

**Acyclic Automata.** In both the upper and the lower bound proof for Theorem 2, we employ the membership problem for acyclic automata, which has already been studied in connection with the knapsack and subset sum problem [8, 16].

We define a *finite automaton* as a tuple  $\mathcal{A} = (Q, \Sigma, \Delta, q_0, q_f)$ , where  $Q$  is a finite set of states,  $\Sigma$  is the *input alphabet*,  $q_0 \in Q$  is the *initial state*,  $q_f \in Q$  is the *final state*, and  $\Delta \subseteq Q \times \Sigma^* \times Q$  is a finite set of *transitions*. The language accepted by  $\mathcal{A}$  is denoted with  $L(\mathcal{A})$ . An *acyclic automaton* is a finite automaton  $\mathcal{A} = (Q, \Sigma, \Delta, q_0, q_f)$  such that the relation  $\{(p, q) \mid \exists w \in \Sigma^* : (p, w, q) \in \Delta\}$  is acyclic. For a graph group  $\mathbb{G}(A, I)$  the *membership problem for acyclic automata* is the following computational problem:

**Input:** An acyclic automaton  $\mathcal{A}$  over the input alphabet  $A \cup A^{-1}$ .

**Question:** Is there a word  $w \in L(\mathcal{A})$  such that  $w = 1$  in  $\mathbb{G}(A, I)$ ?

In order to show the upper bound in Theorem 2, we reduce knapsack for  $\mathbb{G}(A, I)$  with  $(A, I)$  a transitive forest to the membership problem for acyclic automata for  $\mathbb{G}(A, I)$  (note that for subset sum this reduction is obvious). Then, we apply Proposition 3 below. From work of Frenkel, Nikolaev, and Ushakov [8], it follows that the membership problem for acyclic automata is in P. We strengthen this to LogCFL by constructing inductively over the operations of free product and direct product with  $\mathbb{Z}$  a logspace-bounded auxiliary pushdown automaton working in polynomial time (these machines accept exactly the languages in LogCFL) that checks whether an acyclic automaton accepts a word that is trivial in  $\mathbb{G}(A, I)$ .

► **Proposition 3.** *If  $(A, I)$  is a transitive forest, then the membership problem for acyclic automata over  $\mathbb{G}(A, I)$  is in LogCFL.*

## 4.1 Bounds on knapsack solutions

As mentioned above, we reduce for graph groups  $\mathbb{G}(A, I)$  with  $(A, I)$  a transitive forest the knapsack problem to the membership problem for acyclic automata. To this end, we show that every solvable knapsack instance has a solution where all exponents are bounded polynomially in the size of the instance. The latter is the most involved proof in our paper.

Frenkel, Nikolaev, and Ushakov [8] call groups with this property *polynomially bounded knapsack groups* and show that this class is closed under taking free products. However, it is not clear if direct products with  $\mathbb{Z}$  also inherit this property and we leave this question open.

Hence, we are looking for a property that yields polynomial size solutions and is passed on to free products and to direct products with  $\mathbb{Z}$ . It is known that the solution sets are always semilinear. If  $(A, I)$  is a transitive forest, this follows from a more general semilinearity property of rational sets [17] and for arbitrary graph groups, this was shown in [19]. Note that it is not true that the solution sets always have polynomial size semilinear representations. This already fails in the case of  $\mathbb{Z}$ : The equation  $x_1 + \dots + x_k = k$  has  $\binom{2k-1}{k} \geq 2^k$  solutions. We will show here that the solution sets have semilinear representations where every occurring number is bounded by a polynomial.

For a vector  $x = (x_1, \dots, x_k) \in \mathbb{Z}^k$ , we define the norm  $\|x\| = \max\{|x_i| \mid i \in [1, k]\}$ . For a subset  $T \subseteq \mathbb{N}^k$ , we write  $T^\oplus$  for the smallest subset of  $\mathbb{N}^k$  that contains  $\{0\} \cup T$  and is closed under addition. A subset  $S \subseteq \mathbb{N}^k$  is called *linear* if there is a vector  $x \in \mathbb{N}^k$  and a finite set  $F \subseteq \mathbb{N}^k$  such that  $S = x + F^\oplus$ . Note that a set is linear if and only if it can be written as  $x + AN^t$  for some  $x \in \mathbb{N}^k$  and some matrix  $A \in \mathbb{N}^{k \times t}$ . Here,  $AN^t$  denotes the set of all vectors  $Ay$  for  $y \in N^t$ . A *semilinear set* is a finite union of linear sets. If  $S = \bigcup_{i=1}^n x_i + F_i^\oplus$  for  $x_1, \dots, x_n \in \mathbb{N}^k$  and finite sets  $F_1, \dots, F_n \subseteq \mathbb{N}^k$ , then the tuple  $(x_1, F_1, \dots, x_n, F_n)$  is a *semilinear representation* of  $S$  and the *magnitude* of this representation is defined as the maximum of  $\|y\|$ , where  $y$  ranges over all elements of  $\bigcup_{i=1}^n \{x_i\} \cup F_i$ . The *magnitude* of a semilinear set  $S$  is the smallest magnitude of a semilinear representation for  $S$ .

► **Definition 4.** A group  $G$  is called *knapsack tame* if there is a polynomial  $p$  such that for every exponent equation  $h_0 g_1^{x_1} h_1 g_2^{x_2} h_2 \dots g_n^{x_n} h_n = 1$  of size  $n$  with pairwise distinct variables  $x_1, \dots, x_k$ , the set  $S \subseteq \mathbb{N}^k$  of solutions is semilinear of magnitude at most  $p(n)$ .

Observe that although the size of an exponent equation may depend on the chosen generating set of  $G$ , changing the generating set increases the size only by a constant factor. Thus, whether or not a group is knapsack tame is independent of the chosen generating set.

► **Theorem 5.** *If  $(A, I)$  is a transitive forest, then  $\mathbb{G}(A, I)$  is knapsack tame.*

Note that Theorem 5 implies in particular that every solvable exponent equation has a polynomially bounded solution. Theorem 5 and Proposition 3 easily yield the upper bound in Theorem 2.

We prove Theorem 5 by showing that knapsack tameness transfers from groups  $G$  to  $G \times \mathbb{Z}$  (Proposition 6) and from  $G$  and  $H$  to  $G * H$  (Proposition 10). Since the trivial group is obviously knapsack tame, the inductive characterization of groups  $\mathbb{G}(A, I)$  for transitive forests  $(A, I)$  immediately yields Theorem 5.

## 4.2 Tameness of direct products with $\mathbb{Z}$

In this section, we sketch a proof of the following result.

► **Proposition 6.** *If  $G$  is knapsack tame, then so is  $G \times \mathbb{Z}$ .*

**Linear Diophantine equations.** We employ a result of Pottier [23], which bounds the norm of minimal non-negative solutions to a linear Diophantine equation. Let  $A \in \mathbb{Z}^{k \times m}$  be an integer matrix where  $a_{ij}$  is the entry of  $A$  at row  $i$  and column  $j$ . We will use the norms  $\|A\|_{1, \infty} = \max_{i \in [1, k]} (\sum_{j \in [1, m]} |a_{ij}|)$ ,  $\|A\|_{\infty, 1} = \max_{j \in [1, m]} (\sum_{i \in [1, k]} |a_{ij}|)$  and  $\|A\|_\infty = \max_{i \in [1, k], j \in [1, m]} |a_{ij}|$  for matrices and  $\|x\|_1 = \sum_{i=1}^m |x_i|$  for vectors  $x \in \mathbb{Z}^m$ . Recall that  $\|x\| = \max_{i \in [1, m]} |x_i|$ . A solution  $x \in \mathbb{N}^m \setminus \{0\}$  to the equation  $Ax = 0$  is *minimal* if there is no  $y \in \mathbb{N}^m \setminus \{0\}$  with  $Ay = 0$  and  $y \leq x$ ,  $y \neq x$ . The set of all solutions clearly forms a submonoid of  $\mathbb{N}^m$ . Let  $r$  be the rank of  $A$ .

► **Theorem 7** (Pottier [23]). *Each non-trivial minimal solution  $x \in \mathbb{N}^m$  to  $Ax = 0$  satisfies  $\|x\|_1 \leq (1 + \|A\|_{1,\infty})^r$ .*

By applying Theorem 7 to the matrix  $(A \mid -b)$ , it is easy to deduce that for each  $x \in \mathbb{N}^m$  with  $Ax = b$ , there is a  $y \in \mathbb{N}^m$  with  $Ay = b$ ,  $y \leq x$ , and  $\|y\|_1 \leq (1 + \|(A \mid -b)\|_{1,\infty})^{r+1}$ . We reformulate Theorem 7 as follows.

► **Lemma 8.** *If  $B \in \mathbb{Z}^{\ell \times k}$  has rank  $r$  and  $b \in \mathbb{Z}^\ell$ , then there exist  $c_1, \dots, c_s \in \mathbb{N}^k$ ,  $C \in \mathbb{N}^{k \times t}$  with  $\|c_i\|_1, \|C\|_{\infty,1} \leq (1 + \|B\|_{1,\infty} + \|b\|)^{r+1}$  such that  $\{x \in \mathbb{N}^k \mid Bx = b\} = \{c_1, \dots, c_s\} + C\mathbb{N}^t$ .*

We want to apply Lemma 8 in a situation where we have no bound on  $\|B\|_{1,\infty}$ , but only one on  $\|B\|_\infty$ . However, we will know that  $\ell = 1$ , which allows us to bound magnitudes in terms of  $\|B\|_\infty$  in the following lemma. Then, Proposition 6 is straightforward to show.

► **Lemma 9.** *If  $B \in \mathbb{Z}^{1 \times k}$  and  $b \in \mathbb{Z}$  with  $\|B\|_\infty, |b| \leq M$ , then we have a decomposition  $\{x \in \mathbb{N}^k \mid Bx = b\} = \{c_1, \dots, c_s\} + C\mathbb{N}^t$  where  $\|c_i\|_1$  and  $\|C\|_{\infty,1}$  are at most  $(M + 1)^4$ .*

### 4.3 Tameness of free products

This section is devoted to the proof of the following Proposition.

► **Proposition 10.** *If  $G_0$  and  $G_1$  are knapsack tame, then so is  $G_0 * G_1$ .*

Let  $G = G_0 * G_1$ . Suppose that for  $i \in \{0, 1\}$  the group  $G_i$  is generated by  $A_i$ , where  $A_i^{-1} = A_i$  and let  $A = A_0 \uplus A_1$ . Recall that every  $g \in G$  can be written uniquely as  $g = g_1 \cdots g_n$  where  $g_i \in (G_0 \setminus \{1\}) \cup (G_1 \setminus \{1\})$  for each  $i \in [1, n]$  and where  $g_j \in G_t$  iff  $g_{j+1} \in G_{1-t}$  for  $j \in [1, n - 1]$ . We call  $g$  *cyclically reduced* if either  $n \in \{0, 1\}$  or  $n \geq 2$  and for some  $t \in \{0, 1\}$ , either  $g_1 \in G_t$  and  $g_n \in G_{1-t}$  or  $g_1, g_n \in G_t$  and  $g_n g_1 \neq 1$ .

Every word  $w \in A^*$  has a (possibly empty) unique factorization into maximal factors from  $A_0^+ \cup A_1^+$ , which we call *syllables*. By  $\|w\|$ , we denote the number of syllables of  $w$ . The word  $w$  is *reduced* if none of its syllables represents 1 (in  $G_0$  resp.  $G_1$ ). We define the maps  $\lambda, \rho: A^+ \rightarrow A^+$  ("rotate left/right"), where for each word  $w \in A^+$  with its factorization  $w = w_1 \cdots w_m$  into syllables, we set  $\lambda(w) = w_2 \cdots w_m w_1$  and  $\rho(w) = w_m w_1 w_2 \cdots w_{m-1}$ .

Consider a word  $w = w_1 \cdots w_m \in A^*$ , where for each  $i \in [1, m]$ , we have  $w_i \in A_j^+$  for some  $j \in \{0, 1\}$  (we allow  $w_i, w_{i+1} \in A_j^+$ ). A *cancellation* is a subset  $C \subseteq 2^{[1, m]}$  that is

- *a partition:*  $\bigcup_{I \in C} I = [1, m]$  and  $I \cap J = \emptyset$  for any  $I, J \in C$  with  $I \neq J$ .
- *consistent:* for each  $I \in C$ , there is an  $i \in \{0, 1\}$  such that  $w_j \in A_i^*$  for all  $j \in I$ .
- *cancelling:* if  $\{i_1, \dots, i_\ell\} \in C$  with  $i_1 < \dots < i_\ell$ , then  $w_{i_1} \cdots w_{i_\ell}$  represents 1 in  $G$ .
- *well-nested:* there are no  $I, J \in C$  with  $i_1, i_2 \in I$ ,  $j_1, j_2 \in J$  and  $i_1 < j_1 < i_2 < j_2$ .
- *maximal:* if  $w_i, w_{i+1} \in A_j^+$  for  $j \in \{0, 1\}$  then there is an  $I \in C$  with  $i, i + 1 \in I$ .

Since  $C$  can be regarded as a hypergraph on  $[1, m]$ , the elements of  $C$  will be called *edges*. It is not hard to show that a word  $w$  admits a cancellation if and only if  $w = 1$  in  $G$ .

Consider now an exponent equation

$$h_0 g_1^{x_1} h_1 \cdots g_k^{x_k} h_k = 1, \tag{1}$$

of size  $n$ , where  $g_i$  is represented by  $u_i \in A^*$  for  $i \in [1, k]$  and  $h_i$  is represented by  $v_i \in A^*$  for  $i \in [0, k]$ . Then clearly  $\sum_{i=0}^k |v_i| + \sum_{i=1}^k |u_i| \leq n$ . Let  $S \subseteq \mathbb{N}^k$  be the set of all solutions to (1). Of course, when showing that  $S$  has a polynomial magnitude, we may assume that  $g_i \neq 1$  for any  $i \in [1, k]$ . Moreover, we lose no generality by assuming that all words  $u_i$ ,  $i \in [1, k]$  and  $v_i$ ,  $i \in [0, k]$  are reduced. Furthermore, we may assume that each  $g_i$  is cyclically reduced. Indeed, if some  $g_i$  is not cyclically reduced, we can write  $g_i = f^{-1} g f$  for some cyclically

reduced  $g$  and replace  $h_{i-1}$ ,  $g_i$ , and  $h_i$  by  $h_{i-1}f^{-1}$ ,  $g = fg_i f^{-1}$ , and  $fh_i$ , respectively. This does not change the solution set because  $h_{i-1}f^{-1}(fg_i f^{-1})^{x_i}fh_i = h_{i-1}g_i^{x_i}h_i$ . Moreover, if we do this replacement for each  $g_i$  that is not cyclically reduced, we increase the size of the instance by at most  $2|g_1| + \dots + 2|g_k| \leq 2n$  (note that  $|g| = |g_i|$ ). Applying this argument again, we may even assume that  $u_i \in A_0^+ \cup A_1^+ \cup A_0^+ A^* A_1^+ \cup A_1^+ A^* A_0^+$  for every  $i \in [1, k]$ . Note that  $\lambda$  and  $\rho$  are bijections on words of this form.

Consider a solution  $(x_1, \dots, x_k)$  to (1). Then the word

$$w = v_0 u_1^{x_1} v_1 \cdots u_k^{x_k} v_k \quad (2)$$

represents 1 in  $G = G_0 * G_1$ . We factorize each  $v_i$ ,  $i \in [0, k]$ , and each  $u_i$ ,  $i \in [1, k]$ , into its syllables. These factorizations define a factorization  $w = w_1 \cdots w_m$  and we call this the *block factorization* of  $w$ . This is the coarsest refinement of the factorization  $w = v_0 u_1^{x_1} v_1 \cdots u_k^{x_k} v_k$  and of  $w$ 's factorization into syllables. The numbers  $1, 2, \dots, m$  are the *blocks* of  $w$ . We fix this factorization  $w = w_1 \cdots w_m$  for the rest of this section.

**Certified solutions.** In the representation  $v_0 u_1^{x_1} v_1 \cdots u_k^{x_k} v_k = 1$  of (1), the words  $u_1, \dots, u_k$  are called the *cycles*. If  $u_i \in A_0^+ \cup A_1^+$ , the cycle  $u_i$  is said to be *simple* and otherwise *mixed* (note that  $u_i = \varepsilon$  cannot happen because  $g_i \neq 1$ ). Let  $p$  be a block of  $w$ . If  $w_p$  is contained in some  $u_i^{x_i}$  for a cycle  $u_i$ , then  $p$  is a  $u_i$ -*block* or *block from*  $u_i$ . If  $w_p$  is contained in some  $v_i$ , then  $p$  is a  $v_i$ -*block* or a *block from*  $v_i$ . A *certified solution* is a pair  $(x, C)$ , where  $x$  is a solution to (1) and  $C$  is a cancellation of the word  $w$  as in (2). An edge  $I \in C$  is called *standard* if  $|I| = 2$  and the two blocks in  $I$  are from mixed cycles. Intuitively, the following tells us that in a cancellation, most edges are standard.

► **Lemma 11.** *Let  $C$  be a cancellation and  $u_i$  be a mixed cycle. Then there are at most  $n + 3k + 1$  non-standard edges  $I \in C$  containing a  $u_i$ -block.*

**Mixed periods.** From now on, for each  $i \in [1, k]$ , we use  $e_i$  to denote the  $i$ -th unit vector in  $\mathbb{N}^k$ , i.e. the vector with 1 in the  $i$ -th coordinate and 0 otherwise. A *mixed period* is a vector  $\pi \in \mathbb{N}^k$  of the form  $\|u_j\| \cdot e_i + \|u_i\| \cdot e_j$ , where  $u_i$  and  $u_j$  are mixed cycles. Let  $\mathbb{P} \subseteq \mathbb{N}^k$  be the set of mixed periods. Note that  $|\mathbb{P}| \leq k^2$ .

We will need a condition that guarantees that a given period  $\pi \in \mathbb{P}$  can be added to a solution  $x$  to obtain another solution. Suppose we have two blocks  $p$  and  $q$  for which we know that if we insert a string  $f_1$  to the left of  $w_p$  and a string  $f_2$  to the right of  $w_q$  and  $f_1 f_2$  cancels to 1 in  $G$ , then the whole word cancels to 1. Which string would we insert to the left of  $w_p$  and to the right of  $w_q$  if we build the solution  $x + \pi$ ?

Suppose  $p$  is a  $u_i$ -block and  $q$  is a  $u_j$ -block. Moreover, let  $r$  be the first (left-most)  $u_i$ -block and let  $s$  be the last (right-most)  $u_j$ -block. If we add  $\|u_j\| \cdot e_i$  to  $x$ , this inserts  $\lambda^{p-r}(u_i^{\|u_j\|})$  to the left of  $w_p$ : Indeed, in the case  $p = r$ , we insert  $u_i^{\|u_j\|}$ ; and when  $p$  moves one position to the right, the inserted string is rotated once to the left. Similarly, if we add  $\|u_i\| \cdot e_j$  to  $x$ , we insert  $\rho^{s-q}(u_j^{\|u_i\|})$  to the right of  $w_q$ : This is clear for  $q = s$  and decrementing  $q$  means rotating the inserted string to the right. This motivates the following definition:

Let  $(x, C)$  be a certified solution and let  $u_i$  and  $u_j$  be mixed cycles with  $i < j$ . Moreover, let  $r \in [1, m]$  be the left-most  $u_i$ -block and let  $s \in [1, m]$  be the right-most  $u_j$ -block. Then the mixed period  $\pi = \|u_j\| \cdot e_i + \|u_i\| \cdot e_j$  is *compatible with*  $(x, C)$  if there are a  $u_i$ -block  $p$  and a  $u_j$ -block  $q$  such that

$$\{p, q\} \in C \text{ and } \lambda^{p-r}(u_i^{\|u_j\|})\rho^{s-q}(u_j^{\|u_i\|}) \text{ represents 1 in } G. \quad (3)$$



With  $\mathbb{P}(x, C)$ , we denote the set of mixed periods that are compatible with  $(x, C)$ . One might wonder why we require an edge  $\{p, q\} \in C$ . In order to guarantee that  $\lambda^{p-r}(u_i^{\|u_j\|})$  and  $\rho^{s-q}(u_j^{\|u_i\|})$  can cancel, it would be sufficient to merely forbid edges  $I \in C$  that intersect  $[p, q]$  and contain a block outside of  $[p-1, q+1]$ . However, this weaker condition can become false when we insert other mixed periods. Our stronger condition is preserved, which implies:

► **Lemma 12.** *Let  $(x, C)$  be a certified solution. Then every  $x' \in x + \mathbb{P}(x, C)^\oplus$  is a solution.*

Let  $M \subseteq [1, k]$  be the set of  $i \in [1, k]$  such that  $u_i$  is a mixed cycle and  $\|x\|_m = \max_{i \in M} x_i$ .

► **Lemma 13.** *There is a polynomial  $q$  such that the following holds. For every certified solution  $(x, C)$  with  $\|x\|_m > q(n)$ , there exists a mixed period  $\pi \in \mathbb{P}$  and a certified solution  $(x', C')$  such that  $x = x' + \pi$ ,  $\pi \in \mathbb{P}(x', C')$ , and  $\mathbb{P}(x, C) \subseteq \mathbb{P}(x', C')$ .*

**Proof.** We show that the lemma holds if  $q(n) \geq (n + 3k + 1) + kn^2$ . (Recall that  $k \leq n$ .) Let  $(x, C)$  be a certified solution with  $\|x\|_m > q(n)$ . Then there is a mixed cycle  $u_i$  such that  $x_i > q(n)$  and hence  $u_i^{x_i}$  consists of more than  $q(n)$  blocks. Let  $D \subseteq C$  be the set of all edges  $I \in C$  that contain a block from  $u_i$ . It is not hard to show that an edge can contain at most one block per mixed cycle. Hence, we have  $|D| > q(n)$  and, by Lemma 11,  $D$  contains more than  $kn^2$  standard edges. Therefore, there must exist a mixed cycle  $u_j$  such that the set  $E \subseteq D$  of standard edges  $I \in D$  that consist of one block from  $u_i$  and one block from  $u_j$  satisfies  $|E| > n^2$ . Let  $B_i$  (resp.,  $B_j$ ) be the set of blocks from  $u_i$  (resp.,  $u_j$ ) contained in some edge  $I \in E$ . One can show that  $B_i$  and  $B_j$  are intervals of size more than  $n^2$ .

We only deal with the case  $i < j$ , the case  $i > j$  can be done similarly. Let us take a subinterval  $[p', p]$  of  $B_i$  such that  $p - p' = \|u_i\| \cdot \|u_j\| \leq n^2$ . By well-nestedness and since  $B_j$  is an interval, the neighbors (with respect to the edges from  $E$ ) of  $[p', p]$  form an interval  $[q, q'] \subseteq B_j$  as well, and we have  $p - p' = q' - q = \|u_i\| \cdot \|u_j\|$ . Moreover, we have an edge  $\{p - \ell, q + \ell\} \in E$  for each  $\ell \in [0, p - p']$ . In particular,  $w_{p'} w_{p'+1} \cdots w_{p-1} w_{q+1} \cdots w_{q'}$  represents 1 in  $G$ .

Let  $r$  be the left-most  $u_i$ -block and let  $s$  be the right-most  $u_j$ -block. Then, as shown before the definition of compatibility, we have

$$\lambda^{p-r}(u_i^{\|u_j\|}) = w_{p'} w_{p'+1} \cdots w_{p-1} \text{ and } \rho^{s-q}(u_j^{\|u_i\|}) = w_{q+1} w_{q+1} \cdots w_{q'}.$$

Therefore,  $\lambda^{p-r}(u_i^{\|u_j\|}) \rho^{s-q}(u_j^{\|u_i\|})$  represents 1 in  $G$  and  $\{p, q\}$  witnesses compatibility of  $\pi = \|u_j\| \cdot e_i + \|u_i\| \cdot e_j$  with  $(x, C)$ . Hence,  $\pi \in \mathbb{P}(x, C)$ .

Let  $x' = x - \pi$ . We remove the factors  $w_{p'} \cdots w_{p-1}$  and  $w_{q+1} \cdots w_{q'}$  from  $w$ . Then, the remaining blocks spell  $w' = v_0 u_1^{x'_1} v_1 \cdots u_k^{x'_k} v_k$ . Indeed, recall that removing from a word  $y^t$  any factor of length  $\ell \cdot |y|$  will result in the word  $y^{t-\ell}$ . Moreover, let  $C'$  be the set of edges that agree with  $C$  on the remaining blocks. By the choice of the removed blocks, it is clear that  $C'$  is a cancellation for  $w'$ . Hence,  $(x', C')$  is a certified solution.

It remains to verify  $\mathbb{P}(x, C) \subseteq \mathbb{P}(x', C')$ . First note that for every mixed cycle  $u_\ell$ , all  $u_\ell$ -blocks that remain in  $w'$  change their position relative to the left-most and the right-most  $u_\ell$ -block by a difference that is divisible by  $\|u_\ell\|$  (if  $i \neq \ell \neq j$  then these relative positions do not change at all). Note that the expression  $\lambda^{p-r}(u_i^{\|u_j\|})$  is not altered when  $p - r$  changes by a difference divisible by  $\|u_i\|$ , and an analogous fact holds for  $\rho^{s-q}(u_j^{\|u_i\|})$ . Hence, the edge in  $C'$  that corresponds to the  $C$ -edge  $\{p, q\}$  is a witness for  $\pi \in \mathbb{P}(x', C')$ . Moreover, for all other mixed periods  $\pi' \in \mathbb{P}(x, C) \setminus \{\pi\}$  that are witnessed by an edge  $\{t, u\} \in C$ , the blocks  $t$  and  $u$  do not belong to  $[p', p-1] \cup [q+1, q']$ . Therefore, the corresponding edge in  $C'$  exists and serves as a witness for  $\pi' \in \mathbb{P}(x', C')$ . ◀

Repeated application of Lemma 13 now yields:

► **Lemma 14.** *There exists a polynomial  $q$  such that the following holds. For every solution  $x \in \mathbb{N}^k$ , there exists a certified solution  $(x', C')$  such that  $\|x'\|_m \leq q(n)$  and  $x \in x' + \mathbb{P}(x', C')^\oplus$ .*

We are now ready to prove Proposition 10 and thus Theorem 5.

**Proof of Proposition 10.** Suppose that  $p_0$  and  $p_1$  are the polynomials guaranteed by the knapsack tameness of  $G_0$  and  $G_1$ , respectively. Recall that  $S \subseteq \mathbb{N}^k$  is the set of solutions to (1). We prove that there exists a polynomial  $p$  such that for every  $x \in S$  there is a semilinear set  $S' \subseteq \mathbb{N}^k$  of magnitude at most  $p(n)$  such that  $x \in S' \subseteq S$ . This clearly implies that  $S$  has magnitude at most  $p(n)$ . First, we apply Lemma 14. It yields a polynomial  $q$  and a certified solution  $(x', C')$  with  $\|x'\|_m \leq q(n)$  such that  $x \in x' + \mathbb{P}(x', C')^\oplus$ . Let  $w' = v_0 u_1^{x'_1} v_1 \cdots u_k^{x'_k} v_k$  and consider  $w'$  decomposed into blocks as we did above with  $w$ .

Let  $T \subseteq [1, k]$  be the set of all  $i \in [1, k]$  for which the cycle  $u_i$  is simple. Since  $C'$  is maximal, for each  $i \in T$ , all  $u_i$ -blocks are contained in one edge  $I_i \in C'$ . Note that an edge may contain the blocks of more than one simple cycle. We partition  $T$  into sets  $T = T_1 \uplus \cdots \uplus T_t$  so that  $i \in T$  and  $j \in T$  belong to the same part if and only if the  $u_i$ -blocks and the  $u_j$ -blocks belong to the same edge of  $C$ , i.e.  $I_i = I_j$ .

For a moment, let us fix an  $\ell \in [1, t]$  and let  $I \in C'$  be the edge containing all  $u_i$ -blocks for all the  $i \in T_\ell$ . Moreover, let  $T_\ell = \{i_1, \dots, i_r\}$ . The words  $\bar{v}_j$  for  $j \in [0, r]$  will collect those blocks that belong to  $I$  but are not  $u_{i_s}$ -blocks for any  $s \in [1, r]$ . Formally,  $\bar{v}_0$  consists of all blocks that belong to  $I$  that are to the left of all  $u_{i_1}$ -blocks. Similarly,  $\bar{v}_r$  is the concatenation of all blocks belonging to  $I$  that are to the right of all  $u_{i_r}$ -blocks. Finally, for  $j \in [1, r-1]$ ,  $\bar{v}_j$  consists of all blocks that belong to  $I$  and are to the right of all  $u_{i_j}$ -blocks and to the left of all  $u_{i_{j+1}}$ -blocks. By consistency of  $C'$ , for some  $s \in \{0, 1\}$ , all the words  $\bar{v}_j$  for  $j \in [0, r]$  and the words  $u_{i_j}$  for  $j \in [1, r]$  belong to  $A_s^*$  and thus represent elements of  $G_s$ . Since  $G_s$  is knapsack tame, we know that the set

$$S_\ell = \{z \in \mathbb{N}^k \mid \bar{v}_0 u_{i_1}^{z_{i_1}} \bar{v}_1 u_{i_2}^{z_{i_2}} \bar{v}_2 \cdots u_{i_r}^{z_{i_r}} \bar{v}_r \text{ represents } 1 \text{ in } G_s, \quad z_j = 0 \text{ for } j \notin T_\ell\}$$

has magnitude at most  $p_s(n)$ . Consider the vector  $y \in \mathbb{N}^k$  with  $y_i = 0$  for  $i \in T$  and  $y_i = x'_i$  for  $i \in [1, k] \setminus T$  (i.e. when  $u_i$  is a mixed cycle). We claim that  $S' = y + S_1 + \cdots + S_t + \mathbb{P}(x', C')^\oplus$  has magnitude at most  $q(n) + p_0(n) + p_1(n) + n$  and satisfies  $x \in S' \subseteq S$ .

First, since  $y$  and the members of  $S_1, \dots, S_t$  are non-zero on pairwise disjoint coordinates, the magnitude of  $y + S_1 + \cdots + S_t$  is the maximum of  $\|y\|$  and the maximal magnitude of  $S_1, \dots, S_t$ . Hence, it is bounded by  $q(n) + p_0(n) + p_1(n)$ . The summand  $\mathbb{P}(x', C')^\oplus$  contributes only periods, and their magnitude is bounded by  $n$  (recall that they are mixed periods). Thus, the magnitude of  $S'$  is at most  $p(n) = q(n) + p_0(n) + p_1(n) + n$ .

The cancelling property of  $(x', C')$  tells us that  $x' - y$  is contained in  $S_1 + \cdots + S_t$ . By the choice of  $(x', C')$ , we have  $x \in x' + \mathbb{P}(x', C')^\oplus$ . Together, this means  $x \in S'$ . Hence, it remains to show  $S' \subseteq S$ . To this end, consider a vector  $x'' \in y + S_1 + \cdots + S_t$ . It differs from  $x'$  only in the exponents at simple cycles. Therefore, we can apply essentially the same cancellation to  $x''$  as to  $x'$ : we just need to adjust the edges containing the blocks of simple cycles. It is therefore clear that the resulting cancellation  $C''$  has the same compatible mixed periods as  $C'$ :  $\mathbb{P}(x'', C'') = \mathbb{P}(x', C')$ . Thus, by Lemma 12, we have  $x'' + \mathbb{P}(x', C')^\oplus \subseteq S$ . This proves  $S' = y + S_1 + \cdots + S_t + \mathbb{P}(x', C')^\oplus \subseteq S$  and hence Proposition 10. ◀

#### 4.4 LogCFL-hardness

It remains to show the lower bound in Theorem 2. If  $(A, I)$  is not complete, then  $(A, I)$  contains two non-adjacent vertices and thus  $\mathbb{G}(A, I)$  contains an isomorphic copy of  $F_2$ , the

free group of rank two. Hence, we will show that knapsack and subset sum for  $F_2$  are LogCFL-hard. Let  $\{a, b\}$  be a generating set for  $F_2$ . Let  $\theta: \{a, b, a^{-1}, b^{-1}\}^* \rightarrow F_2$  be the morphism that maps a word  $w$  to the group element represented by  $w$ . A *valence automaton* over a group  $G$  is a tuple  $\mathcal{A} = (Q, \Sigma, \Delta, q_0, q_f)$  where  $Q, \Sigma, q_0, q_f$  are as in a finite automaton and  $\Delta$  is a finite subset of  $Q \times \Sigma^* \times G \times Q$ . The *language accepted by  $\mathcal{A}$*  is denoted  $L(\mathcal{A})$  and consists of all words  $w_1 \cdots w_n$  such that there is a computation  $p_0 \xrightarrow{w_1, g_1} p_1 \rightarrow \cdots \rightarrow p_{n-1} \xrightarrow{w_n, g_n} p_n$  such that  $(p_{i-1}, w_i, g_i, p_i) \in \Delta$  for  $i \in [1, n]$  and  $p_0 = q_0, p_n = q_f$ , and  $g_1 \cdots g_n = 1$  in  $G$ .

Fix a context-free language  $L \subseteq \Sigma^*$  with a LogCFL-complete membership problem; such languages exist [10]. The Chomsky-Schützenberger theorem implies that there exists a valence automaton  $\mathcal{A}$  over  $F_2$  such that  $L = L(\mathcal{A})$ . Moreover, analyzing the proof of the Chomsky-Schützenberger theorem from [14] shows that there exists a constant  $c$  such that for every  $w \in \Sigma^*$  we have:  $w \in L(\mathcal{A}) = L$  if and only if there exists an accepting run of  $\mathcal{A}$  for  $w$  of length at most  $c \cdot |w|$ . Given the word  $w \in \Sigma$ , it is easy to convert the valence automaton  $\mathcal{A}$  into an acyclic automaton over  $\{a, b, a^{-1}, b^{-1}\}^*$  that exhausts all computations of  $\mathcal{A}$  of length at most  $c \cdot |w|$ . This yields the following:

► **Proposition 15.** *For  $F_2$ , the membership problem for acyclic automata is LogCFL-hard.*

► **Proposition 16.** *For  $F_2$ , knapsack and subset sum are LogCFL-hard.*

**Proof.** Let  $\mathcal{A} = (Q, \{a, b, a^{-1}, b^{-1}\}, \Delta, q_0, q_f)$  be an acyclic automaton. We construct words  $w, w_1, \dots, w_m \in \{a, b, a^{-1}, b^{-1}\}^*$  such that  $1 \in \theta(L(\mathcal{A}))$  if and only if  $\theta(w) \in \theta(w_1^* w_2^* \cdots w_m^*)$  if and only if  $\theta(w) \in \theta(w_1^{e_1} w_2^{e_2} \cdots w_m^{e_m})$  for some  $e_1, e_2, \dots, e_m \in \{0, 1\}$ . W.l.o.g. assume that  $Q = \{1, \dots, n\}$ , where 1 is the initial state and  $n$  is the unique final state of  $\mathcal{A}$ .

Let  $\alpha_i = a^i b a^{-i}$  for  $i \in [1, n+2]$ . It is well known that the  $\alpha_i$  generate a free subgroup of rank  $n+2$  in  $F_2$  [20, Proposition 3.1]. Define the embedding  $\varphi: F_2 \rightarrow F_2$  by  $\varphi(a) = \alpha_{n+1}$  and  $\varphi(b) = \alpha_{n+2}$ . For a transition  $t = (p, w, q) \in \Delta$  let  $\tilde{t} = \alpha_p \varphi(w) \alpha_q^{-1}$ . Let  $\tilde{\Delta} = \{t_1, \dots, t_m\}$  such that  $t_i = (p, a, q)$  and  $t_j = (q, b, r)$  implies  $i < j$ . Since  $\mathcal{A}$  is acyclic, such an enumeration must exist. Together with the fact that the  $\alpha_i$  generate a free group, it follows that  $1 \in \theta(L(\mathcal{A}))$  if and only if  $\theta(\alpha_1 \alpha_n^{-1}) \in \theta(\tilde{t}_1^* \tilde{t}_2^* \cdots \tilde{t}_m^*)$  if and only if  $\theta(\alpha_1 \alpha_n^{-1}) \in \theta(\tilde{t}_1^{e_1} \tilde{t}_2^{e_2} \cdots \tilde{t}_m^{e_m})$  for some  $e_1, e_2, \dots, e_m \in \{0, 1\}$ . ◀

## 5 NP-completeness

In [19], the authors proved that knapsack for the graph group  $\mathbb{G}(\text{C4}) \cong F_2 \times F_2$  is NP-complete. Here we extend this result to all graph groups  $\mathbb{G}(A, I)$  where  $(A, I)$  is not a transitive forest. An *acyclic loop automaton* is a finite automaton  $\mathcal{A} = (Q, \Sigma, \Delta, q_0, q_f)$  such that there exists a linear order  $\preceq$  on  $\Delta$  having the property that for all  $(p, u, q), (q, v, r) \in \Delta$  it holds  $(p, u, q) \preceq (q, v, r)$ . Thus, an acyclic loop automaton is obtained from an acyclic automaton by attaching to some of the states a unique loop. For a trace monoid  $\mathbb{M}(A, I)$ , *intersection nonemptiness for acyclic loop automata* is the following computational problem:

**Input:** Two acyclic loop automata  $\mathcal{A}_1, \mathcal{A}_2$  over the input alphabet  $A$ .

**Question:** Does  $[L(\mathcal{A}_1)]_I \cap [L(\mathcal{A}_2)]_I \neq \emptyset$  hold?

Aalbersberg and Hoogeboom [1] proved that for the trace monoid  $\mathbb{M}(\text{P4})$ , intersection nonemptiness for arbitrary finite automata is undecidable. We use their technique to show:

► **Lemma 17.** *For  $\mathbb{M}(\text{P4})$ , intersection nonemptiness for acyclic loop automata is NP-hard.*

**Proof.** We give a reduction from 3SAT. Let  $\varphi = \bigwedge_{i=1}^m C_i$  where for every  $i \in [1, m]$ ,  $C_i = (L_{i,1} \vee L_{i,2} \vee L_{i,3})$  is a clause consisting of three literals. Let  $x_1, \dots, x_n$  be the boolean variables that occur in  $\varphi$ . Every literal  $L_{i,j}$  belongs to  $\{x_1, \dots, x_n, \neg x_1, \dots, \neg x_n\}$ .

Let  $p_1, p_2, \dots, p_n$  be a list of the first  $n$  prime numbers. So, for each boolean variable  $x_i$  we have the corresponding prime number  $p_i$ . We encode a valuation  $\beta: \{x_1, \dots, x_n\} \rightarrow \{0, 1\}$  by any natural number  $N$  such that  $N \equiv 0 \pmod{p_i}$  if and only if  $\beta(x_i) = 1$ . For a positive literal  $x_i$  let  $S(x_i) = \{p_i \cdot n \mid n \in \mathbb{N}\}$  and for a negative literal  $\neg x_i$  let  $S(\neg x_i) = \{p_i \cdot n + r \mid n \in \mathbb{N}, r \in [1, p_i - 1]\}$ . Moreover, for every  $i \in [1, m]$  let  $S_i = S(L_{i,1}) \cup S(L_{i,2}) \cup S(L_{i,3})$ . Thus,  $S_i$  is the set of all numbers that encode a valuation that makes the clause  $C_i$  true. Hence, the set  $S = \bigcap_{i=1}^m S_i$  encodes the set of all valuations that make  $\varphi$  true.

We first construct an acyclic loop automaton  $\mathcal{A}_1$  with  $L(\mathcal{A}_1) = \prod_{i=1}^m \{a(bc)^{N_i}d \mid N_i \in S_i\}$ . Note that  $\varphi$  is satisfiable iff  $[L(\mathcal{A}_1)]_I$  contains a trace from  $[\{(a(bc)^N d)^m \mid N \in \mathbb{N}\}]_I$ . We will ensure this property with a second acyclic loop automaton  $\mathcal{A}_2$  that satisfies the equality  $L(\mathcal{A}_2) = b^*(ad(bc)^*)^{m-1}adc^*$ . We claim that  $[L(\mathcal{A}_1)]_I \cap [L(\mathcal{A}_2)]_I = [\{(a(bc)^N d)^m \mid N \in S\}]_I$ .

First assume that  $w \equiv_I (a(bc)^N d)^m$  for some  $N \in S$ . We have  $w \equiv_I (a(bc)^N d)^m \equiv_I b^N (ad(bc)^N)^{m-1} adc^N$  and thus  $[w]_I \in [L(\mathcal{A}_2)]_I$ . Moreover, since  $N \in S$  we get  $[w]_I \in [L(\mathcal{A}_1)]_I$ . For the other direction, let  $[w]_I \in [L(\mathcal{A}_1)]_I \cap [L(\mathcal{A}_2)]_I$ . Thus

$$w \equiv_I \prod_{i=1}^m (a(bc)^{N_i}d) \equiv_I b^{N_1} \left( \prod_{i=1}^{m-1} adc^{N_i} b^{N_{i+1}} \right) adc^{N_m},$$

where  $N_i \in S_i$  for  $i \in [1, m]$ . Moreover,  $[w]_I \in [L(\mathcal{A}_2)]_I$  yields  $k_0, k_1, \dots, k_{m-1}, k_m \geq 0$  with

$$b^{N_1} \left( \prod_{i=1}^{m-1} adc^{N_i} b^{N_{i+1}} \right) adc^{N_m} \equiv_I b^{k_0} \left( \prod_{i=1}^{m-1} ad(bc)^{k_i} \right) adc^{k_m} \equiv_I b^{k_0} \left( \prod_{i=1}^{m-1} (adb^{k_i} c^{k_i}) \right) adc^{k_m}.$$

Since every symbol depends on  $a$  or on  $d$ , this identity implies  $N_i = N_{i+1}$  for  $i \in [1, m-1]$ . Thus,  $[w]_I \in [\{(a(bc)^N d)^m \mid N \in S\}]_I$ .  $\blacktriangleleft$

For a graph group  $\mathbb{G}(A, I)$  the *membership problem for acyclic loop automata* is the following computational problem:

**Input:** An acyclic loop automaton  $\mathcal{A}$  over the input alphabet  $A \cup A^{-1}$ .

**Question:** Is there a word  $w \in L(\mathcal{A})$  such that  $w = 1$  in  $\mathbb{G}(A, I)$ ?

It is straightforward to reduce the intersection emptiness problem for acyclic loop automata over  $\mathbb{M}(A, I)$  to the membership problem for acyclic loop automata over  $\mathbb{G}(A, I)$ .

► **Lemma 18.** *For  $\mathbb{G}(P4)$ , the membership problem for acyclic loop automata is NP-hard.*

We can now use a construction from [17] to reduce membership for acyclic loop automata over  $\mathbb{G}(P4)$  to knapsack for  $\mathbb{G}(P4)$ .

► **Lemma 19.** *Knapsack for the graph group  $\mathbb{G}(P4)$  is NP-hard.*

► **Theorem 20.** *If  $(A, I)$  is an independence alphabet, which is not a transitive forest, then knapsack for the graph group  $\mathbb{G}(A, I)$  is NP-complete.*

**Proof.** If  $(A, I)$  is not a transitive forest, then  $P4$  or  $C4$  is an induced subgraph of  $(A, I)$  [27]. Thus,  $\mathbb{G}(P4)$  or  $\mathbb{G}(C4) \cong F_2 \times F_2$  is a subgroup of  $\mathbb{G}(A, I)$ . Hence, NP-hardness of knapsack for  $\mathbb{G}(A, I)$  follows from [19] or Lemma 19.  $\blacktriangleleft$

## 6 An open problem

In [19] the authors proved that (uncompressed) subset sum for  $\mathbb{G}(C4)$  is NP-complete as well. It remains open whether subset sum is NP-hard also for  $\mathbb{G}(P4)$ . Our proof for the NP-hardness of knapsack for  $\mathbb{G}(P4)$  makes essential use of exponentially large exponents and hence cannot be used for subset sum.

---

**References**

---

- 1 I. J. Aalbersberg and H. J. Hoogeboom. Characterizations of the decidability of some problems for regular trace languages. *Mathematical Systems Theory*, 22:1–19, 1989.
- 2 I. Agol. The virtual Haken conjecture. With an appendix by Agol, Daniel Groves, and Jason Manning. *Documenta Mathematica*, 18:1045–1087, 2013.
- 3 L. Babai, R. Beals, J. Cai, G. Ivanyos, and E. M. Luks. Multiplicative equations over commuting matrices. In *Proceedings of SODA 1996*, pages 498–507. ACM/SIAM, 1996.
- 4 M. Bestvina and N. Brady. Morse theory and finiteness properties of groups. *Inventiones Mathematicae*, 129(3):445–470, 1997.
- 5 J. Crisp and B. Wiest. Embeddings of graph braid and surface groups in right-angled Artin groups and braid groups. *Algebraic & Geometric Topology*, 4:439–472, 2004.
- 6 V. Diekert. *Combinatorics on Traces*, volume 454 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.
- 7 M. Elberfeld, A. Jakoby, and T. Tantau. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:128, 2011.
- 8 E. Frenkel, A. Nikolaev, and A. Ushakov. Knapsack problems in products of groups. *Journal of Symbolic Computation*, 74:96–108, 2016.
- 9 R. Ghrist and V. Peterson. The geometry and topology of reconfiguration. *Advances in Applied Mathematics*, 38(3):302–323, 2007.
- 10 S. Greibach. The hardest context-free language. *SIAM Journal on Computing*, 2(4):304–310, 1973.
- 11 C. Haase. *On the complexity of model checking counter automata*. PhD thesis, University of Oxford, St Catherine’s College, 2011.
- 12 F. Haglund and D. T. Wise. Coxeter groups are virtually special. *Advances in Mathematics*, 224(5):1890–1903, 2010.
- 13 B. Jenner. Knapsack problems for NL. *Information Processing Letters*, 54(3):169–174, 1995.
- 14 M. Kambites. Formal languages and groups as memory. *Communications in Algebra*, 37:193–208, 2009. doi:10.1080/00927870802243580.
- 15 R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.
- 16 D. König, M. Lohrey, and G. Zetsche. 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.
- 17 M. Lohrey and B. Steinberg. The submonoid and rational subset membership problems for graph groups. *Journal of Algebra*, 320(2):728–755, 2008.
- 18 M. Lohrey and G. Zetsche. The complexity of knapsack in graph groups. Technical report, arXiv.org, 2015. URL: <https://arxiv.org/abs/1610.00373>.
- 19 M. Lohrey and G. Zetsche. Knapsack in graph groups, HNN-extensions and amalgamated products. In *Proceedings of STACS 2016*, volume 47 of *LIPICs*, pages 50:1–50:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.
- 20 R. C. Lyndon and P. E. Schupp. *Combinatorial Group Theory*. Springer-Verlag, 1977.
- 21 A. Myasnikov, A. Nikolaev, and A. Ushakov. Knapsack problems in groups. *Mathematics of Computation*, 84:987–1016, 2015.
- 22 C. H. Papadimitriou. On the complexity of integer programming. *Journal of the Association for Computing Machinery*, 28(4):765–768, 1981.

## 52:14 The Complexity of Knapsack in Graph Groups

- 23 L. Pottier. Minimal solutions of linear diophantine systems : bounds and algorithms. In *Proceedings of RTA 1991*, volume 488 of *Lecture Notes in Computer Science*, pages 162–173. Springer-Verlag, 1991.
- 24 I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25(3):405–414, 1978.
- 25 H. Vollmer. *Introduction to Circuit Complexity*. Springer-Verlag, 1999.
- 26 D. T. Wise. Research announcement: the structure of groups with a quasiconvex hierarchy. *Electronic Research Announcements in Mathematical Sciences*, 16:44–55, 2009.
- 27 E. S. Wolk. A note on “The comparability graph of a tree”. *Proceedings of the American Mathematical Society*, 16:17–20, 1965.

# Algorithmic Information, Plane Kakeya Sets, and Conditional Dimension

Jack H. Lutz<sup>\*1</sup> and Neil Lutz<sup>†2</sup>

1 Department of Computer Science, Iowa State University, Ames, IA, USA  
lutz@cs.iastate.edu

2 Department of Computer Science, Rutgers University, Piscataway, NJ, USA  
njlutz@rutgers.edu

---

## Abstract

We formulate the *conditional Kolmogorov complexity of  $x$  given  $y$*  at precision  $r$ , where  $x$  and  $y$  are points in Euclidean spaces and  $r$  is a natural number. We demonstrate the utility of this notion in two ways.

1. We prove a *point-to-set principle* that enables one to use the (relativized, constructive) dimension of a *single point* in a set  $E$  in a Euclidean space to establish a lower bound on the (classical) Hausdorff dimension of  $E$ . We then use this principle, together with conditional Kolmogorov complexity in Euclidean spaces, to give a new proof of the known, two-dimensional case of the Kakeya conjecture. This theorem of geometric measure theory, proved by Davies in 1971, says that every plane set containing a unit line segment in every direction has Hausdorff dimension 2.
2. We use conditional Kolmogorov complexity in Euclidean spaces to develop the *lower* and *upper conditional dimensions*  $\dim(x|y)$  and  $\text{Dim}(x|y)$  of  $x$  given  $y$ , where  $x$  and  $y$  are points in Euclidean spaces. Intuitively these are the lower and upper asymptotic algorithmic information densities of  $x$  conditioned on the information in  $y$ . We prove that these conditional dimensions are robust and that they have the correct information-theoretic relationships with the well-studied dimensions  $\dim(x)$  and  $\text{Dim}(x)$  and the mutual dimensions  $\text{mdim}(x : y)$  and  $\text{Mdim}(x : y)$ .

**1998 ACM Subject Classification** F.1.3 Complexity Measures and Classes

**Keywords and phrases** algorithmic randomness, conditional dimension, geometric measure theory, Kakeya sets, Kolmogorov complexity

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.53

## 1 Introduction

This paper concerns the fine-scale geometry of algorithmic information in Euclidean spaces. It shows how new ideas in algorithmic information theory can shed new light on old problems in geometric measure theory. This introduction explains these new ideas, a general principle for applying these ideas to classical problems, and an example of such an application. It also describes a newer concept in algorithmic information theory that arises naturally from this work.

---

\* Research supported in part by National Science Foundation Grants 1247051 and 1545028.

† This work was conducted at DIMACS and at Hebrew University. It was partially enabled through support from the National Science Foundation under grants CCF-1445755 and CCF-1101690.



© Jack H. Lutz and Neil Lutz;  
licensed under Creative Commons License CC-BY

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 53; pp. 53:1–53:13

Leibniz International Proceedings in Informatics



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

Roughly fifteen years after the mid-twentieth century development of the *Shannon information theory* of probability spaces [29], Kolmogorov recognized that Turing’s mathematical theory of computation could be used to refine the Shannon theory to enable the amount of information in individual data objects to be quantified [17]. The resulting theory of *Kolmogorov complexity*, or *algorithmic information theory*, is now a large enterprise with many applications in computer science, mathematics, and other sciences [20]. Kolmogorov proved the first version of the fundamental relationship between the Shannon and algorithmic theories of information in [17], and this relationship was made exquisitely precise by Levin’s coding theorem [18, 19]. (Solomonoff and Chaitin independently developed Kolmogorov complexity at around the same time as Kolmogorov with somewhat different motivations [30, 6, 7].)

At the turn of the present century, the first author recognized that Hausdorff’s 1919 theory of fractal dimension [16] is an older theory of information that can also be refined using Turing’s mathematical theory of computation, thereby enabling the *density* of information in individual infinite data objects, such as infinite binary sequences or points in Euclidean spaces, to be quantified [21, 22]. The resulting theory of *effective fractal dimensions* is now an active enterprise with a growing array of applications [11]. The paper [22] proved a relationship between effective fractal dimensions and Kolmogorov complexity that is as precise as – and uses – Levin’s coding theorem.

Most of the work on effective fractal dimensions to date has concerned the (*constructive*) *dimension*  $\dim(x)$  and the dual *strong (constructive) dimension*  $\text{Dim}(x)$  [1] of an infinite data object  $x$ , which for purposes of the present paper is a point in a Euclidean space  $\mathbb{R}^n$  for some positive integer  $n$ .<sup>1</sup> The inequalities

$$0 \leq \dim(x) \leq \text{Dim}(x) \leq n$$

hold generally, with, for example,  $\text{Dim}(x) = 0$  for points  $x$  that are computable and  $\dim(x) = n$  for points that are algorithmically random in the sense of Martin-Löf [Mart66].

How can the dimensions of individual points – dimensions that are defined using the theory of computing – have any bearing on classical problems of geometric measure theory? The problems that we have in mind here are problems in which one seeks to establish lower bounds on the classical Hausdorff dimensions  $\dim_H(E)$  (or other fractal dimensions) of sets  $E$  in Euclidean spaces. Such problems involve global properties of sets and make no mention of algorithms.

The key to bridging this gap is relativization. Specifically, we prove here a *point-to-set principle* saying that, in order to prove a lower bound  $\dim_H(E) \geq \alpha$ , it suffices to show that, for every  $A \subseteq \mathbb{N}$  and every  $\varepsilon > 0$ , there is a point  $x \in E$  such that  $\dim^A(x) \geq \alpha - \varepsilon$ , where  $\dim^A(x)$  is the dimension of  $x$  relative to the oracle  $A$ . We also prove the analogous point-to-set principle for the classical packing dimension  $\dim_P(E)$  and the relativized strong dimension  $\text{Dim}^A(x)$ .

We illustrate the power of the point-to-set principle by using it to give a new proof of a known theorem in geometric measure theory. A Kakeya set in a Euclidean space  $\mathbb{R}^n$  is a set  $K \subseteq \mathbb{R}^n$  that contains a unit line segment in every direction. Besicovitch [2, 3] proved that Kakeya sets can have Lebesgue measure 0 and asked whether Kakeya sets in the Euclidean plane can have dimension less than 2 [9]. The famous Kakeya conjecture asserts a negative answer to this and to the analogous question in higher dimensions, i.e., states that every

---

<sup>1</sup> These constructive dimensions are  $\Sigma_1^0$  effectivizations of Hausdorff and packing dimensions [13]. Other effectivizations, e.g., computable dimensions, polynomial time dimensions, and finite-state dimensions, have been investigated, but only the constructive dimensions are discussed here.



Keakeya set in a Euclidean space  $\mathbb{R}^n$  has Hausdorff dimension  $n$ .<sup>2</sup> This conjecture holds trivially for  $n = 1$  and was proven by Davies [9] for  $n = 2$ . A version of the conjecture in finite fields has been proven by Dvir [12]. For Euclidean spaces of dimension  $n \geq 3$ , it is an important open problem with deep connections to other problems in analysis [35, 32].

In this paper we use our point-to-set principle to give a new proof of Davies’s theorem. This proof does not resemble the classical proof, which is not difficult but relies on Marstrand’s projection theorem [26] and point-line duality. Instead of analyzing the set  $K$  globally, our proof focuses on the information content of a single, judiciously chosen point in  $K$ . Given a Keakeya set  $K \subseteq \mathbb{R}^2$  and an oracle  $A \subseteq \mathbb{N}$ , we first choose a particular line segment  $L \subseteq K$  and a particular point  $(x, mx+b) \in L$ , where  $y = mx+b$  is the equation of the line containing  $L$ .<sup>3</sup> We then show that  $\dim^A(x, mx+b) \geq 2$ . By our point-to-set principle this implies that  $\dim_H(K) \geq 2$ .

Our proof that  $\dim^A(x, mx+b) \geq 2$  requires us to formulate a concept of conditional Kolmogorov complexity in Euclidean spaces. Specifically, for points  $x \in \mathbb{R}^m$  and  $y \in \mathbb{R}^n$  and natural numbers  $r$ , we develop the *conditional Kolmogorov complexity*  $K_r(x|y)$  of  $x$  given  $y$  at *precision*  $r$ . This is a “conditional version” of the Kolmogorov complexity  $K_r(x)$  of  $x$  at precision  $r$  that has been used in several recent papers (e.g., [24, 4, 15]).

In addition to enabling our new proof of Davies’s theorem, conditional Kolmogorov complexity in Euclidean spaces enables us to fill a gap in effective dimension theory. The fundamental quantities in Shannon information theory are the *entropy* (information content)  $H(X)$  of a probability space  $X$ , the *conditional entropy*  $H(X|Y)$  of a probability space  $X$  given a probability space  $Y$ , and the *mutual information* (shared information)  $I(X; Y)$  between two probability spaces  $X$  and  $Y$  [8]. The analogous quantities in Kolmogorov complexity theory are the *Kolmogorov complexity*  $K(u)$  of a finite data object  $u$ , the *conditional Kolmogorov complexity*  $K(u|v)$  of a finite data object  $u$  given a finite data object  $v$ , and the *algorithmic mutual information*  $I(u : v)$  between two finite data objects  $u$  and  $v$  [20]. The above-described dimensions  $\dim(x)$  and  $\text{Dim}(x)$  of a point  $x$  in Euclidean space (or an infinite sequence  $x$  over a finite alphabet) are analogous by limit theorems [27, 1] to  $K(u)$  and hence to  $H(X)$ . Case and the first author have recently developed and investigated the *mutual dimension*  $\text{mdim}(x : y)$  and the dual *strong mutual dimension*  $\text{Mdim}(x : y)$ , which are densities of the algorithmic information shared by points  $x$  and  $y$  in Euclidean spaces [4] or sequences  $x$  and  $y$  over a finite alphabet [5]. These mutual dimensions are analogous to  $I(u : v)$  and  $I(X; Y)$ .

What is conspicuously missing from the above account is a notion of conditional dimension. In this paper we remedy this by using conditional Kolmogorov complexity in Euclidean space to develop the *conditional dimension*  $\dim(x|y)$  of  $x$  given  $y$  and its dual, the *conditional strong dimension*  $\text{Dim}(x|y)$  of  $x$  given  $y$ , where  $x$  and  $y$  are points in Euclidean spaces. We prove that these conditional dimensions are well behaved and that they have the correct information theoretic relationships with the previously defined dimensions and mutual dimensions. The original plan of our proof of Davies’s theorem used conditional dimensions, and we developed their basic theory to that end. Our final proof of Davies’s theorem does not use them, but conditional dimensions (like the conditional entropy and conditional Kolmogorov complexity that motivate them) are very likely to be useful in future investigations.

<sup>2</sup> Statements of the Keakeya conjecture vary in the literature. For example, the set is sometimes required to be compact or Borel, and the dimension used may be Minkowski instead of Hausdorff. Since the Hausdorff dimension of a set is never greater than its Minkowski dimension, our formulation is at least as strong as those variations.

<sup>3</sup> One might naïvely expect that for independently random  $m$  and  $x$ , the point  $(x, mx+b)$  must be random. In fact, in every direction there is a line that contains no random point [23].

The rest of this paper is organized as follows. Section 2 briefly reviews the dimensions of points in Euclidean spaces. Section 3 presents the point-to-set principles that enable us to use dimensions of individual points to prove lower bounds on classical fractal dimensions. Section 4 develops conditional Kolmogorov complexity in Euclidean spaces. Section 5 uses the preceding two sections to give our new proof of Davies's theorem. Section 6 uses Section 4 to develop conditional dimensions in Euclidean spaces.

## 2 Dimensions of Points in Euclidean Spaces

This section reviews the constructive notions of dimension and mutual dimension in Euclidean spaces. The presentation here is in terms of Kolmogorov complexity. Briefly, the *conditional Kolmogorov complexity*  $K(w|v)$  of a string  $w \in \{0,1\}^*$  given a string  $v \in \{0,1\}^*$  is the minimum length  $|\pi|$  of a binary string  $\pi$  for which  $U(\pi, v) = w$ , where  $U$  is a fixed universal self-delimiting Turing machine. The *Kolmogorov complexity* of  $w$  is  $K(w|\lambda)$ , where  $\lambda$  is the empty string. We write  $U(\pi)$  for  $U(\pi, \lambda)$ . When  $U(\pi) = w$ , the string  $\pi$  is called a *program* for  $w$ . The quantity  $K(w)$  is also called the *algorithmic information content* of  $w$ . Routine coding extends this definition from  $\{0,1\}^*$  to other discrete domains, so that the Kolmogorov complexities of natural numbers, rational numbers, tuples of these, etc., are well defined up to additive constants. Detailed discussions of self-delimiting Turing machines and Kolmogorov complexity appear in the books [20, 28, 11] and many papers.

The definition of  $K(q)$  for rational points  $q$  in Euclidean space is lifted in two steps to define the dimensions of arbitrary points in Euclidean space. First, for  $x \in \mathbb{R}^n$  and  $r \in \mathbb{N}$ , the *Kolmogorov complexity* of  $x$  at *precision*  $r$  is

$$K_r(x) = \min\{K(q) : q \in \mathbb{Q}^n \cap B_{2^{-r}}(x)\}, \quad (2.1)$$

where  $B_{2^{-r}}(x)$  is the open ball with radius  $2^{-r}$  and center  $x$ . Second, for  $x \in \mathbb{R}^n$ , the *dimension* and *strong dimension* of  $x$  are

$$\dim(x) = \liminf_{r \rightarrow \infty} \frac{K_r(x)}{r} \quad \text{and} \quad \text{Dim}(x) = \limsup_{r \rightarrow \infty} \frac{K_r(x)}{r}, \quad (2.2)$$

respectively.<sup>4</sup>

Intuitively,  $\dim(x)$  and  $\text{Dim}(x)$  are the lower and upper asymptotic densities of the algorithmic information in  $x$ . These quantities were first defined in Cantor spaces using betting strategies called gales and shown to be constructive versions of classical Hausdorff and packing dimension, respectively [22, 1]. These definitions were explicitly extended to Euclidean spaces in [24], where the identities (2.2) were *proven as a theorem*. Here it is convenient to use these identities as definitions. For  $x \in \mathbb{R}^n$ , it is easy to see that

$$0 \leq \dim(x) \leq \text{Dim}(x) \leq n,$$

and it is known that, for any two reals  $0 \leq \alpha \leq \beta \leq n$ , there exist uncountably many points  $x \in \mathbb{R}^n$  satisfying  $\dim(x) = \alpha$  and  $\text{Dim}(x) = \beta$  [1]. Applications of these dimensions in Euclidean spaces appear in [24, 14, 25, 10, 15].

<sup>4</sup> We note that  $K_r(x) = K(x \upharpoonright r) + o(r)$ , where  $x \upharpoonright r$  is the binary expansion of  $x$ , truncated  $r$  bits to the right of the binary point. However, it has been known since Turing's famous correction [33] that binary notation is not a suitable representation for the arguments and values of computable functions on the reals. (See also [34].) Hence, in order to make our definitions useful for further work in computable analysis, we formulate complexities and dimensions in terms of rational approximations, both here and later.

### 3 From Points to Sets

The central message of this paper is a useful *point-to-set principle* by which the existence of a single high-dimensional point in a set  $E \subseteq \mathbb{R}^n$  implies that the set  $E$  has high dimension.

To formulate this principle we use relativization. All the algorithmic information concepts in Sections 2 and 6 above can be relativized to an arbitrary oracle  $A \subseteq \mathbb{N}$  by giving the Turing machine in their definitions oracle access to  $A$ . Relativized Kolmogorov complexity  $K_r^A(x)$  and relativized dimensions  $\dim^A(x)$  and  $\text{Dim}^A(x)$  are thus well defined. Moreover, the results of Section 2 hold relative to any oracle  $A$ .

We first establish the point-to-set principle for Hausdorff dimension. Let  $E \subseteq \mathbb{R}^n$ . For  $\delta > 0$ , define  $\mathcal{U}_\delta(E)$  to be the collection of all countable covers of  $E$  by sets of positive diameter at most  $\delta$ . That is, for every cover  $\{U_i\}_{i \in \mathbb{N}} \in \mathcal{U}_\delta(E)$ , we have  $E \subseteq \bigcup_{i \in \mathbb{N}} U_i$  and  $|U_i| \in (0, \delta]$  for all  $i \in \mathbb{N}$ , where for  $X \in \mathbb{R}^n$ ,  $|X| = \sup_{p, q \in X} |p - q|$ . For  $s \geq 0$ , define

$$H_\delta^s(E) = \inf \left\{ \sum_{i \in \mathbb{N}} |U_i|^s : \{U_i\}_{i \in \mathbb{N}} \in \mathcal{U}_\delta(E) \right\}.$$

Then the *s-dimensional Hausdorff outer measure* of  $E$  is

$$H^s(E) = \lim_{\delta \rightarrow 0^+} H_\delta^s(E),$$

and the *Hausdorff dimension* of  $E$  is

$$\dim_H(E) = \inf \{s > 0 : H^s(E) = 0\}.$$

More details may be found in standard texts, e.g., [31, 13].

► **Theorem 1.** (Point-to-set principle for Hausdorff dimension) *For every set  $E \subseteq \mathbb{R}^n$ ,*

$$\dim_H(E) = \min_{A \subseteq \mathbb{N}} \sup_{x \in E} \dim^A(x).$$

Three things should be noted about this principle. First, while the left-hand side is the *classical* Hausdorff dimension, which is a global property of  $E$  that does not involve the theory of computing, the right-hand side is a pointwise property of the set that makes essential use of relativized algorithmic information theory. Second, as the proof shows, the right-hand side is a minimum, not merely an infimum. Third, and most crucially, this principle implies that, in order to prove a lower bound  $\dim_H(E) \geq \alpha$ , it suffices to show that, for every  $A \subseteq \mathbb{N}$  and every  $\varepsilon > 0$ , there is a point  $x \in E$  such that  $\dim^A(x) \geq \alpha - \varepsilon$ .<sup>5</sup>

For the ( $\geq$ ) direction of this principle, we construct the minimizing oracle  $A$ . The oracle encodes, for a carefully chosen sequence of increasingly refined covers for  $E$ , the approximate locations and diameters of all cover elements. Using this oracle, a point  $x \in \mathbb{R}^n$  can be approximated by specifying an appropriately small cover element that it belongs to, which requires an amount of information that depends on the number of similarly-sized cover elements. We use the definition of Hausdorff dimension to bound that number. The ( $\leq$ ) direction can be shown using results from [24], but in the interest of self-containment we prove it directly.

<sup>5</sup> The  $\varepsilon$  here is useful in general but is not needed in some cases, including our proof of Theorem 5 below.

**Proof of Theorem 1.** Let  $E \subseteq \mathbb{R}^n$ , and let  $d = \dim_H(E)$ . For every  $s > d$  we have  $H^s(E) = 0$ , so there is a sequence  $\{\{U_i^{t,s}\}_{i \in \mathbb{N}}\}_{t \in \mathbb{N}}$  of countable covers of  $E$  such that  $|U_i^{t,s}| \leq 2^{-t}$  for every  $i, t \in \mathbb{N}$ , and for every sufficiently large  $t$  we have

$$\sum_{i \in \mathbb{N}} |U_i^{t,s}|^s < 1. \quad (3.1)$$

Let  $D = \mathbb{N}^3 \times (\mathbb{Q} \cap (d, \infty))$ . Our oracle  $A$  encodes functions  $f_A : D \rightarrow \mathbb{Q}^n$  and  $g_A : D \rightarrow \mathbb{Q}$  such that for every  $(i, t, r, s) \in D$ , we have

$$f_A(i, t, r, s) \in B_{2^{-r-1}}(u)$$

for some  $u \in U_i^{t,s}$  and

$$\left| g_A(i, t, r, s) - |U_i^{t,s}| \right| < 2^{-r-4}. \quad (3.2)$$

We will show, for every  $x \in E$  and rational  $s > d$ , that  $\dim^A(x) \leq s$ .

Fix  $x \in E$  and  $s \in \mathbb{Q} \cap (d, \infty)$ . If for any  $i_0, t_0 \in \mathbb{N}$  we have  $x \in U_{i_0}^{t_0, s}$  and  $|U_{i_0}^{t_0, s}| = 0$ , then  $U_{i_0}^{t_0, s} = \{x\}$ , so  $f_A(i_0, t_0, r, s) \in B_{2^{-r}}(x)$  for every  $r \in \mathbb{N}$ . In this case, let  $M$  be a prefix Turing machine with oracle access to  $A$  such that, whenever  $U(\iota) = i \in \mathbb{N}$ ,  $U(\tau) = t \in \mathbb{N}$ ,  $U(\rho) = r \in \mathbb{N}$ , and  $U(\sigma) = q \in \mathbb{Q} \cap (d, \infty)$ ,

$$M(\iota\tau\rho\sigma) = f_A(i, t, r, q).$$

Now for any  $r \in \mathbb{N}$ , let  $\iota, \tau, \rho$ , and  $\sigma$  be witnesses to  $K(i_0)$ ,  $K(t_0)$ ,  $K(r)$ , and  $K(s)$ , respectively. Since  $i_0, t_0$ , and  $s$  are all constant in  $r$  and  $|\rho| = o(r)$ , we have  $|\iota\tau\rho\sigma| = o(r)$ . Thus  $K_r^A(x) = o(r)$ , and  $\dim^A(x) = 0$ . Hence assume that every cover element containing  $x$  has positive diameter.

Fix sufficiently large  $t$ , and let  $U_{i_x}^{t,s}$  be some cover element containing  $x$ . Let  $M'$  be a self-delimiting Turing machine with oracle access to  $A$  such that whenever  $U(\kappa) = k \in \mathbb{N}$ ,  $U(\tau) = \ell \in \mathbb{N}$ ,  $U(\rho) = r \in \mathbb{N}$ , and  $U(\sigma) = q \in \mathbb{Q} \cap (d, \infty)$ ,

$$M'(\kappa\tau\rho\sigma) = f_A(p, \ell, r, q),$$

where  $p$  is the  $k^{\text{th}}$  index  $i$  such that  $g_A(i, t, r, q) \geq 2^{-r-3}$ .

Now fix  $r \geq t - 1$  such that

$$|U_{i_x}^{t,s}| \in [2^{-r-2}, 2^{-r-1}).$$

Notice that  $g_A(i_x, t, r, s) \geq 2^{-r-3}$ . Hence there is some  $k$  such that, letting  $\kappa, \tau, \rho$ , and  $\sigma$  be witnesses to  $K(k)$ ,  $K(t)$ ,  $K(r)$ , and  $K(s)$ , respectively,

$$M'(\kappa\tau\rho\sigma) \in B_{2^{-r-1}}(u),$$

for some  $u \in U_{i_x}^{t,s}$ . Because  $|U_{i_x}^{t,s}| < 2^{-r-1}$  and  $x \in U_{i_x}^{t,s}$ , we have

$$M'(\kappa\tau\rho\sigma) \in B_{2^{-r}}(x).$$

Thus

$$K_r^A(x) \leq K(k) + K(t) + K(s) + K(r) + c,$$

where  $c$  is a machine constant for  $M'$ . Since  $s$  is constant in  $r$  and  $t < r$ , this expression is  $K(k) + o(r) \leq \log(k) + o(r)$ . By (3.1), there are fewer than  $2^{(r+4)s}$  indices  $i \in \mathbb{N}$  such that

$$|U_i^{t,s}| \geq 2^{-r-4},$$

hence by (3.2) there are fewer than  $2^{(r+4)s}$  indices  $i \in \mathbb{N}$  such that  $g_A(i, t, r, s) \geq 2^{-r-3}$ , so  $\log(k) < (r+4)s$ . Therefore  $K_r^A(x) \leq rs + o(r)$ .

There are infinitely many such  $r$ , which can be seen by replacing  $t$  above with  $r+2$ . We have shown

$$\dim^A(x) = \liminf_{r \rightarrow \infty} \frac{K_r^A(x)}{r} \leq s,$$

for every rational  $s > d$ , hence  $\dim^A(x) \leq d$ . It follows that

$$\min_{A \subseteq \mathbb{N}} \sup_{x \in E} \dim^A(x) \leq d.$$

For the other direction, assume for contradiction that there is some oracle  $A$  and  $d' < d$  such that

$$\sup_{x \in E} \dim^A(x) = d'.$$

Then for every  $x \in E$ ,  $\dim^A(x) \leq d'$ . Let  $s \in (d', d)$ . For every  $r \in \mathbb{N}$ , define the sets

$$\mathcal{B}_r = \{B_{2^{-r}}(q) : q \in \mathbb{Q} \text{ and } K^A(q) \leq rs\}$$

and

$$\mathcal{W}_r = \bigcup_{k=r}^{\infty} \mathcal{B}_k.$$

There are at most  $2^{ks+1}$  balls in each  $\mathcal{B}_k$ , so for every  $r \in \mathbb{N}$  and  $s' \in (s, d)$ ,

$$\begin{aligned} \sum_{W \in \mathcal{W}_r} |W|^{s'} &= \sum_{k=r}^{\infty} \sum_{W \in \mathcal{B}_k} |W|^{s'} \\ &\leq \sum_{k=r}^{\infty} 2^{ks+1} (2^{1-k})^{s'} \\ &= 2^{1+s'} \cdot \sum_{k=r}^{\infty} 2^{(s-s')k}, \end{aligned}$$

which approaches 0 as  $r \rightarrow \infty$ . As every  $\mathcal{W}_r$  is a cover for  $E$ , we have  $H^{s'}(E) = 0$ , so  $\dim_H(E) \leq s' < d$ , a contradiction.  $\blacktriangleleft$

The *packing dimension*  $\dim_P(E)$  of a set  $E \subseteq \mathbb{R}^n$ , defined in standard texts, e.g., [13], is a dual of Hausdorff dimension satisfying  $\dim_P(E) \geq \dim_H(E)$ , with equality for very “regular” sets  $E$ . We also have the following.

► **Theorem 2.** (Point-to-set principle for packing dimension) *For every set  $E \subseteq \mathbb{R}^n$ ,*

$$\dim_P(E) = \min_{A \subseteq \mathbb{N}} \sup_{x \in E} \text{Dim}^A(x).$$

## 4 Conditional Kolmogorov Complexity in Euclidean Spaces

We now develop the conditional Kolmogorov complexity in Euclidean spaces.

For  $x \in \mathbb{R}^m$ ,  $q \in \mathbb{Q}^n$ , and  $r \in \mathbb{N}$ , the *conditional Kolmogorov complexity* of  $x$  at precision  $r$  given  $q$  is

$$\hat{K}_r(x|q) = \min \{K(p|q) : p \in \mathbb{Q}^m \cap B_{2^{-r}}(x)\}. \quad (4.1)$$

For  $x \in \mathbb{R}^m$ ,  $y \in \mathbb{R}^n$ , and  $r, s \in \mathbb{N}$ , the *conditional Kolmogorov complexity* of  $x$  at precision  $r$  given  $y$  at precision  $s$  is

$$K_{r,s}(x|y) = \max \{\hat{K}_r(x|q) : q \in \mathbb{Q}^n \cap B_{2^{-s}}(y)\}. \quad (4.2)$$

Intuitively, the maximizing argument  $q$  is the point near  $y$  that is least helpful in the task of approximating  $x$ . Note that  $K_{r,s}(x|y)$  is finite, because  $\hat{K}_r(x|q) \leq K_r(x) + O(1)$ . For  $x \in \mathbb{R}^m$ ,  $y \in \mathbb{R}^n$ , and  $r \in \mathbb{N}$ , the *conditional Kolmogorov complexity* of  $x$  given  $y$  at precision  $r$  is

$$K_r(x|y) = K_{r,r}(x|y). \quad (4.3)$$

► **Theorem 3** (Chain rule for  $K_r$ ). *For all  $x \in \mathbb{R}^m$  and  $y \in \mathbb{R}^n$ ,*

$$K_r(x, y) = K_r(x|y) + K_r(y) + o(r).$$

We also consider the Kolmogorov complexity of  $x \in \mathbb{R}^m$  at precision  $r$  relative to  $y \in \mathbb{R}^n$ . Let  $K_r^y(x)$  denote  $K_r^{A_y}(x)$ , where  $A_y \subseteq \mathbb{N}$  encodes the binary expansions of  $y$ 's coordinates. The following lemma reflects the intuition that oracle access to  $y$  is at least as useful as any bounded-precision estimate for  $y$ .

► **Lemma 4.** *For each  $m, n \in \mathbb{N}$  there is a constant  $c \in \mathbb{N}$  such that, for all  $x \in \mathbb{R}^m$ ,  $y \in \mathbb{R}^n$ , and  $r, s \in \mathbb{N}$ ,*

$$K_r^y(x) \leq K_{r,s}(x|y) + K(s) + c.$$

*In particular,  $K_r^y(x) \leq K_r(x|y) + K(r) + c$ .*

## 5 Kakeya Sets in the Plane

This section uses the results of the preceding two sections to give a new proof of the following classical theorem. Recall that a *Kakeya set* in  $\mathbb{R}^n$  is a set containing a unit line segment in every direction.

► **Theorem 5** (Davies [9]). *Every Kakeya set in  $\mathbb{R}^2$  has Hausdorff dimension 2.*

Our new proof of Theorem 5 uses a relativized version of the following lemma.

► **Lemma 6.** *Let  $m \in [0, 1]$  and  $b \in \mathbb{R}$ . Then for almost every  $x \in [0, 1]$ ,*

$$\liminf_{r \rightarrow \infty} \frac{K_r(m, b, x) - K_r(b|m)}{r} \leq \dim(x, mx + b). \quad (5.1)$$

**Proof.** We build a program that takes as input a precision level  $r$ , an approximation  $p$  of  $x$ , an approximation  $q$  of  $mx + b$ , a program  $\pi$  that will approximate  $b$  given an approximation for  $m$ , and a natural number  $h$ . In parallel, the program considers each multiple of  $2^{-r}$  in  $[0, 1]$  as a possible approximate value  $u$  for the slope  $m$ , and it checks whether each such  $u$  is consistent with the program's inputs. If  $u$  is close to  $m$ , then  $\pi(u)$  will be close to  $b$ , so  $up + \pi(u)$  will be close to  $mx + b$ . Any  $u$  that satisfies this condition is considered a "candidate" for approximating  $m$ .

Some of these candidates may be “false positives,” in that there can be values of  $u$  that are far from  $m$  but for which  $up + \pi(u)$  is still close to  $mx + b$ . Thus the program is also given an input  $h$  so that it can choose the correct candidate; it selects the  $h^{\text{th}}$  candidate that arises in its execution. We will show that this  $h$  is often not large enough to significantly affect the total input length.

Formally, let  $M$  be a Turing machine that runs the following algorithm on input  $\rho\pi\sigma\eta$  whenever  $U(\rho) = r \in \mathbb{N}$ ,  $U(\eta) = h \in \mathbb{N}$ , and  $U(\sigma) = (p, q) \in \mathbb{Q}^2$ :

```

candidate := 0
for  $i = 0, 1, \dots, 2^r$ , in parallel:
     $u_i := 2^{-r}i$ 
     $v_i := U(\pi, u_i)$ 
do atomically:
    if  $v_i \in \mathbb{R}$  and  $|u_i p + v_i - q| < 2^{2-r}$ , then candidate := candidate + 1
    if candidate =  $h$ , then return  $(u_i, v_i, p)$  and halt
    
```

Fix  $m \in [0, 1]$  and  $b \in \mathbb{R}$ . For each  $r \in \mathbb{N}$ , let  $m_r = 2^{-r} \lfloor m \cdot 2^r \rfloor$ , and fix  $\pi_r$  testifying to the value of  $\hat{K}_r(b|m_r)$  and  $\sigma_r$  testifying to the value of  $K_r(x, mx + b)$ .

The proof is completed by the four following claims. Intuitively, Claim 7 says that no point in  $B_{2^{-r}}(m)$  gives much less information about  $b$  than  $m_r$  does. Claim 8 states that there is always some value of  $h$  that causes this machine to return the desired output. Claim 9 says that for almost every  $x$ , this value does not grow too quickly with  $r$ , and Claim 10 says that (5.1) holds for every such  $x$ .

► **Claim 7.** For every  $r \in \mathbb{N}$ ,  $K_r(b|m) = \hat{K}_r(b|m_r) + o(r)$ .

► **Claim 8.** For each  $x \in [0, 1]$  and  $r \in \mathbb{N}$ , there exists an  $h \in \mathbb{N}$  such that

$$M(\rho\pi_r\sigma_r\eta) \in B_{2^{1-r}}(m, b, x),$$

where  $U(\rho) = r$  and  $U(\eta) = h$ .

For every  $x \in [0, 1]$  and  $r \in \mathbb{N}$ , define  $h(x, r)$  to be the minimal  $h$  satisfying the conditions of Claim 8.

► **Claim 9.** For almost every  $x \in [0, 1]$ ,  $\log(h(x, r)) = o(r)$ .

► **Claim 10.** For every  $x \in [0, 1]$ , if  $\log(h(x, r)) = o(r)$ , then

$$\liminf_{r \rightarrow \infty} \frac{K_r(m, b, x) - K_r(b|m)}{r} \leq \dim(x, mx + b).$$

The lemma follows immediately from Claims 9 and 10. ◀

**Proof of Theorem 5.** Let  $K$  be a Kakeya set in  $\mathbb{R}^2$ . By Theorem 1, there exists an oracle  $A$  such that  $\dim_H(K) = \sup_{p \in K} \dim^A(p)$ .

Let  $m \in [0, 1]$  such that  $\dim^A(m) = 1$ ; such an  $m$  exists by Theorem 4.5 of [22].  $K$  contains a unit line segment  $L$  of slope  $m$ . Let  $(x_0, y_0)$  be the left endpoint of such a segment. Let  $q \in \mathbb{Q} \cap [x_0, x_0 + 1/8]$ , and let  $L'$  be the unit segment of slope  $m$  whose left endpoint is  $(x_0 - q, y_0)$ . Let  $b = y_1 + qm$ , the  $y$ -intercept of  $L'$ .

By a relativized version of Lemma 6, there is some  $x \in [0, 1/2]$  such that  $\dim^{A, m, b}(x) = 1$  and

$$\liminf_{r \rightarrow \infty} \frac{K_r^A(m, b, x) - K_r^A(b|m)}{r} \leq \dim^A(x, mx + b).$$

(This holds because almost every  $x \in [0, 1/2]$  is algorithmically random relative to  $(A, m, b)$  and hence satisfies  $\dim^{A,m,b}(x) = 1$ .) Fix such an  $x$ , and notice that  $(x, mx + b) \in L'$ . Now applying a relativized version of Theorem 3,

$$\begin{aligned} \dim^A(x, mx + b) &\geq \liminf_{r \rightarrow \infty} \frac{K_r^A(m, b, x) - K_r^A(b|m)}{r} \\ &= \liminf_{r \rightarrow \infty} \frac{K_r^A(m, b, x) - K_r^A(b, m) + K_r^A(m)}{r} \\ &= \liminf_{r \rightarrow \infty} \frac{K_r^A(x|b, m) + K_r^A(m)}{r} \\ &\geq \liminf_{r \rightarrow \infty} \frac{K_r^A(x|b, m)}{r} + \liminf_{r \rightarrow \infty} \frac{K_r^A(m)}{r}. \end{aligned}$$

By Lemma 4,  $K_r^A(x|b, m) \geq K_r^{A,b,m}(x) + o(r)$ , so we have

$$\begin{aligned} \dim^A(x, mx + b) &\geq \liminf_{r \rightarrow \infty} \frac{K_r^{A,b,m}(x)}{r} + \liminf_{r \rightarrow \infty} \frac{K_r^A(m)}{r} \\ &= \dim^{A,b,m}(x) + \dim^A(m), \end{aligned}$$

which is 2 by our choices of  $m$  and  $x$ . Since

$$\dim^A(x, mx + b) = \dim^A(x + q, mx + b),$$

there exists a point  $(x + q, mx + b) \in K$  such that  $\dim^A(x + q, mx + b) \geq 2$ . By Theorem 1, the point-to-set principle for Hausdorff dimension, this completes the proof.  $\blacktriangleleft$

It is natural to ask what prevents us from extending this proof to higher-dimensional Euclidean spaces. The point of failure in a direct extension would be Claim 9 in the proof of Lemma 6. Speaking informally, the problem is that the total number of candidates may grow as  $2^{(n-1)r}$ , meaning that  $\log(h(x, r))$  could be  $\Omega((n-2)r)$  for every  $x$ .

## 6 Conditional Dimensions in Euclidean Spaces

The results of Section 4, which were used in the proof of Theorem 5, also enable us to give robust formulations of conditional dimensions.

For  $x \in \mathbb{R}^m$  and  $y \in \mathbb{R}^n$ , the *lower* and *upper conditional dimensions* of  $x$  given  $y$  are

$$\dim(x|y) = \liminf_{r \rightarrow \infty} \frac{K_r(x|y)}{r} \quad \text{and} \quad \text{Dim}(x|y) = \limsup_{r \rightarrow \infty} \frac{K_r(x|y)}{r}, \quad (6.1)$$

respectively.

The use of the same precision bound  $r$  for both  $x$  and  $y$  in (4.3) makes the definitions (6.1) appear arbitrary and “brittle.” The following theorem shows that this is not the case.

► **Theorem 11.** *Let  $s : \mathbb{N} \rightarrow \mathbb{N}$ . If  $|s(r) - r| = o(r)$ , then, for all  $x \in \mathbb{R}^m$  and  $y \in \mathbb{R}^n$ ,*

$$\dim(x|y) = \liminf_{r \rightarrow \infty} \frac{K_{r,s(r)}(x|y)}{r},$$

and

$$\text{Dim}(x|y) = \limsup_{r \rightarrow \infty} \frac{K_{r,s(r)}(x|y)}{r}.$$



The rest of this section is devoted to showing that our conditional dimensions have the correct information theoretic relationships with the previously developed dimensions and mutual dimensions.

Mutual dimensions were developed very recently, and Kolmogorov complexity was the starting point. The *mutual (algorithmic) information* between two strings  $u, v \in \{0, 1\}^*$  is

$$I(u : v) = K(v) - K(v|u).$$

Again, routine coding extends  $K(u|v)$  and  $I(u : v)$  to other discrete domains. Discussions of  $K(u|v)$ ,  $I(u : v)$ , and the correspondence of  $K(u)$ ,  $K(u|v)$ , and  $I(u : v)$  with Shannon entropy, Shannon conditional entropy, and Shannon mutual information appear in [20].

In parallel with (2.1) and (2.2), Case and J. H. Lutz [4] lifted the definition of  $I(p : q)$  for rational points  $p$  and  $q$  in Euclidean spaces in two steps to define the mutual dimensions between two arbitrary points in (possibly distinct) Euclidean spaces. First, for  $x \in \mathbb{R}^m$ ,  $y \in \mathbb{R}^n$ , and  $r \in \mathbb{N}$ , the *mutual information* between  $x$  and  $y$  at *precision*  $r$  is

$$I_r(x : y) = \min \{ I(p : q) : p \in B_{2^{-r}}(x) \cap \mathbb{Q}^m \text{ and } q \in B_{2^{-r}}(y) \cap \mathbb{Q}^n \}, \quad (6.2)$$

where  $B_{2^{-r}}(x)$  and  $B_{2^{-r}}(y)$  are the open balls of radius  $2^{-r}$  about  $x$  and  $y$  in their respective Euclidean spaces. Second, for  $x \in \mathbb{R}^m$  and  $y \in \mathbb{R}^n$ , the *lower* and *upper mutual dimensions* between  $x$  and  $y$  are

$$\text{mdim}(x : y) = \liminf_{r \rightarrow \infty} \frac{I_r(x : y)}{r} \quad \text{and} \quad \text{Mdim}(x : y) = \limsup_{r \rightarrow \infty} \frac{I_r(x : y)}{r}, \quad (6.3)$$

respectively. Useful properties of these mutual dimensions, especially including data processing inequalities, appear in [4].

► **Lemma 12.** *For all  $x \in \mathbb{R}^m$  and  $y \in \mathbb{R}^n$ ,*

$$I_r(x : y) = K_r(x) - K_r(x|y) + o(r).$$

The following bounds on mutual dimension follow from Lemma 12.

► **Theorem 13.** *For all  $x \in \mathbb{R}^m$  and  $y \in \mathbb{R}^n$ , the following hold.*

1.  $\text{mdim}(x : y) \geq \dim(x) - \text{Dim}(x|y)$ .
2.  $\text{Mdim}(x : y) \leq \text{Dim}(x) - \dim(x|y)$ .

Our final theorem is easily derived from Theorem 3.

► **Theorem 14 (Chain rule for dimension).** *For all  $x \in \mathbb{R}^m$  and  $y \in \mathbb{R}^n$ ,*

$$\begin{aligned} \dim(x) + \dim(y|x) &\leq \dim(x, y) \\ &\leq \dim(x) + \text{Dim}(y|x) \\ &\leq \text{Dim}(x, y) \\ &\leq \text{Dim}(x) + \text{Dim}(y|x). \end{aligned}$$

## 7 Conclusion

This paper shows a new way in which theoretical computer science can be used to answer questions that may appear unrelated to computation. We are hopeful that our new proof of Davies’s theorem will open the way for using constructive fractal dimensions to make new progress in geometric measure theory, and that conditional dimensions will be a useful component of the information theoretic apparatus for studying dimension.

**Acknowledgments.** We thank Eric Allender for useful corrections and three anonymous reviewers of an earlier version of this work for helpful input regarding presentation.

---

**References**


---

- 1 Krishna B. Athreya, John M. Hitchcock, Jack H. Lutz, and Elvira Mayordomo. Effective strong dimension in algorithmic information and computational complexity. *SIAM J. Comput.*, 37(3):671–705, 2007. doi:10.1137/S0097539703446912.
- 2 A. S. Besicovitch. Sur deux questions d’intégrabilité des fonctions. *Journal de la Société de physique et de mathématique de l’Université de Perm*, 2:105–123, 1919.
- 3 A. S. Besicovitch. On Kakeya’s problem and a similar one. *Mathematische Zeitschrift*, 27:312–320, 1928.
- 4 Adam Case and Jack H. Lutz. Mutual dimension. *ACM Transactions on Computation Theory*, 7(3):12, 2015.
- 5 Adam Case and Jack H. Lutz. Mutual dimension and random sequences. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Mathematical Foundations of Computer Science 2015 – 40th International Symposium, MFCS 2015, Milan, Italy, August 24–28, 2015, Proceedings, Part II*, volume 9235 of *Lecture Notes in Computer Science*, pages 199–210. Springer, 2015.
- 6 Gregory J. Chaitin. On the length of programs for computing finite binary sequences. *J. ACM*, 13(4):547–569, 1966.
- 7 Gregory J. Chaitin. On the length of programs for computing finite binary sequences: statistical considerations. *J. ACM*, 16(1):145–159, 1969.
- 8 Thomas R. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, second edition, 2006.
- 9 Roy O. Davies. Some remarks on the Kakeya problem. *Proc. Cambridge Phil. Soc.*, 69:417–421, 1971.
- 10 Randall Dougherty, Jack H. Lutz, R. Daniel Mauldin, and Jason Teutsch. Translating the Cantor set by a random real. *Transactions of the American Mathematical Society*, 366:3027–3041, 2014.
- 11 Rod Downey and Denis Hirschfeldt. *Algorithmic Randomness and Complexity*. Springer-Verlag, 2010.
- 12 Zeev Dvir. On the size of Kakeya sets in finite fields. *J. Amer. Math. Soc.*, 22:1093–1097, 2009.
- 13 Kenneth Falconer. *Fractal Geometry: Mathematical Foundations and Applications*. Wiley, third edition, 2014.
- 14 Xiaoyang Gu, Jack H. Lutz, and Elvira Mayordomo. Points on computable curves. In *FOCS*, pages 469–474. IEEE Computer Society, 2006. doi:10.1109/FOCS.2006.63.
- 15 Xiaoyang Gu, Jack H. Lutz, Elvira Mayordomo, and Philippe Moser. Dimension spectra of random subfractals of self-similar fractals. *Ann. Pure Appl. Logic*, 165(11):1707–1726, 2014.
- 16 Felix Hausdorff. Dimension und äusseres Mass. *Mathematische Annalen*, 79:157–179, 1919.
- 17 Andrei N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1(1):1–7, 1965.
- 18 Leonid A. Levin. On the notion of a random sequence. *Soviet Math Dokl.*, 14(5):1413–1416, 1973.
- 19 Leonid A. Levin. Laws of information conservation (nongrowth) and aspects of the foundation of probability theory. *Problemy Peredachi Informatsii*, 10(3):30–35, 1974.
- 20 Ming Li and Paul M.B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, third edition, 2008.
- 21 Jack H. Lutz. Dimension in complexity classes. *SIAM J. Comput.*, 32(5):1236–1259, 2003.

- 22 Jack H. Lutz. The dimensions of individual strings and sequences. *Inf. Comput.*, 187(1):49–79, 2003.
- 23 Jack H. Lutz and Neil Lutz. Lines missing every random point. *Computability*, 4(2):85–102, 2015.
- 24 Jack H. Lutz and Elvira Mayordomo. Dimensions of points in self-similar fractals. *SIAM J. Comput.*, 38(3):1080–1112, 2008. doi:10.1137/070684689.
- 25 Jack H. Lutz and Klaus Weihrauch. Connectivity properties of dimension level sets. *Mathematical Logic Quarterly*, 54:483–491, 2008.
- 26 John M. Marstrand. Some fundamental geometrical properties of plane sets of fractional dimensions. *Proceedings of the London Mathematical Society*, 4(3):257–302, 1954.
- 27 Elvira Mayordomo. A Kolmogorov complexity characterization of constructive Hausdorff dimension. *Inf. Process. Lett.*, 84(1):1–3, 2002.
- 28 Andre Nies. *Computability and Randomness*. Oxford University Press, Inc., New York, NY, USA, 2009.
- 29 Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3–4):379–423, 623–656, 1948.
- 30 Ray J. Solomonoff. A formal theory of inductive inference. *Information and Control*, 7(1–2):1–22, 224–254, 1964.
- 31 Elias M. Stein and Rami Shakarchi. *Real Analysis: Measure Theory, Integration, and Hilbert Spaces*. Princeton Lectures in Analysis. Princeton University Press, 2005.
- 32 Terence Tao. From rotating needles to stability of waves: emerging connections between combinatorics, analysis, and PDE. *Notices Amer. Math. Soc*, 48:294–303, 2000.
- 33 Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. A correction. *Proceedings of the London Mathematical Society*, 43(2):544–546, 1937.
- 34 Klaus Weihrauch. *Computable Analysis: An Introduction*. Springer, 2000.
- 35 T. Wolff. Recent work connected with the Kakeya problem. *Prospects in Mathematics*, pages 129–162, 1999.



# On the Synchronisation Problem over Cellular Automata

Gaétan Richard

Université de Caen Normandie, UNICAEN, ENSICAEN, CNRS, GREYC, Caen, France

gaetan.richard@unicaen.fr

---

## Abstract

Cellular automata are a discrete, synchronous, and uniform dynamical system that give rise to a wide range of dynamical behaviours. In this paper, we investigate whether this system can achieve synchronisation. We study the cases of classical bi-infinite configurations, periodic configurations, and periodic configurations of prime period. In the two former cases, we prove that only a “degenerated” form of synchronisation – there exists a fix-point – is possible. In the latter case, we give an explicit construction of a cellular automaton for which any periodic configuration of prime period eventually converges to cycle of two uniform configurations. Our construction is based upon sophisticated tools: aperiodic NW-deterministic tilings [7] and partitioned intervals [1].

**1998 ACM Subject Classification** F.1.1 Models of Computation

**Keywords and phrases** cellular automata, dynamical systems, aperiodic tiling, synchronisation

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.54

## Introduction

Complex systems are systems compound of many “simple” components whose interactions give birth to a wide range of complex behaviours. To better grasp mechanisms behind such emerging behaviours, a classical theoretical approach is to reproduce these behaviours on a regular and simple model. Introduced to model self-replication [10], cellular automata are an example of such a basic simple discrete dynamical system. They consist in a bi-infinite line of *cells* endowed with a *state* chosen among a finite alphabet. The system evolves thanks to the uniform and synchronous application of a *local rule*. This rule gives the new state of a cell according to its previous state and the ones of its neighbours. Despite its apparent simplicity, cellular automata can exhibit a wide range of complex behaviours [11]. In this paper, we focus on one specific behaviour: *synchronisation*, as in biological cell synchronisation.

Section 1 is devoted to present the context, provide a formal definition, and give several preliminary properties. Inspired by biological cell synchronisation, synchronisation is the following: is it possible, starting from any configuration, to ensure that all cells eventually enter the same state at the same moment? This notion has also strong connections with the Firing Squad Synchronisation Problem [2, 8]. In this paper, the two main specificities are that the system is fully deterministic (no randomness is provided) and we require synchronisation for every configuration.

Results are presented in two parts: in Section 2, we prove that our framework does not allow “real” synchronisation neither over the whole configuration space nor over periodic configurations. Then, in Section 3, we present our main result: a detailed construction of a cellular automaton that – over periodic configurations of prime period – always converges toward a cycle consisting of two uniform configurations, solving the *synchronisation problem* as studied in [5]. Finally, Section 4 gives some consequences of our main result.



© Gaétan Richard;

licensed under Creative Commons License CC-BY

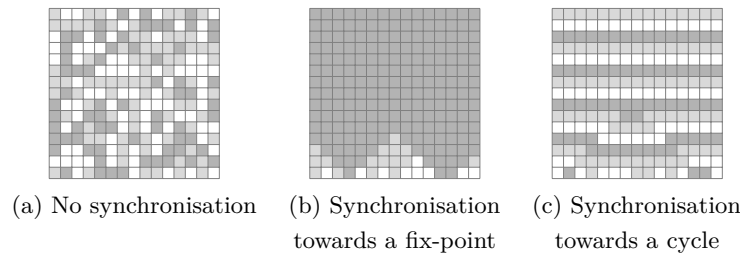
34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 54; pp. 54:1–54:13

Leibniz International Proceedings in Informatics



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



■ **Figure 1** Examples of partial space-time diagrams (time goes up).

## 1 Synchronisation

A *cellular automaton* (CA for short) is a pair  $(Q, f)$  where  $Q$  is a finite set of *states* and  $f : Q^3 \rightarrow Q$  is the *local rule*. The cellular automaton acts on elements of  $Q^{\mathbb{Z}}$  called *configurations*. The resulting action is called *global function*  $F : Q^{\mathbb{Z}} \rightarrow Q^{\mathbb{Z}}$  and is defined by, for any  $c \in Q^{\mathbb{Z}}$ ,  $i \in \mathbb{Z}$ ,  $F(c)_i = f(c_{i-1}, c_i, c_{i+1})$ . The local function is extended to finite words  $f : Q^n \rightarrow Q^{n-2}$  by, for any  $w_0 w_1 \dots w_{n-1} \in Q^n$ ,  $f(w_0 w_1 \dots w_{n-1}) = f(w_0, w_1, w_2) f(w_1, w_2, w_3) \dots f(w_{n-3}, w_{n-2}, w_{n-1})$ . A configuration  $c \in Q^{\mathbb{Z}}$  is of *period*  $p \in \mathbb{N}^+$  if, for any  $i \in \mathbb{Z}$ ,  $c_{i+p} = c_i$ . Due to uniformity of global function, the image of a periodic configuration is periodic and its period divides the original period. In the rest of the paper, the periodic configuration made by the repetition of the non-empty word  $u \in Q^*$  is denoted as  ${}^\omega u^\omega$ . When the period is 1, the configuration is called *uniform*.

The evolution of a configuration  $c \in Q^{\mathbb{Z}}$  is often depicted by piling up the successive iterations  $(c, F(c), F^2(c), \dots)$ . Such a representation is called *space-time diagram*; examples of such a representation can be seen in Figure 1. Intuitively, synchronisation is achieved when, at some time, all cells reach an agreement, i.e. a uniform configuration (see Figure 1).

► **Definition 1** (Synchronisation). A cellular automaton  $(Q, f)$  is *synchronizing* if, for any configuration  $c \in Q^{\mathbb{Z}}$ , there exist  $n_c \in \mathbb{N}^*$  and  $a \in Q$ , such that  $F^{n_c}(c) = {}^\omega a^\omega$ .

Moreover, the cellular automaton is *fully synchronizing* if  $a$  does not depend on  $c$  – i.e., there exists  $a \in Q$  such that, for any configuration  $c \in Q^{\mathbb{Z}}$ , there exists  $n_c \in \mathbb{N}^*$  such that  $F^{n_c}(c) = {}^\omega a^\omega$ .

A careful reader may notice that our definition allows cases where the synchronisation does not bring meaningful information: for example, when a configuration converges to a uniform fix-point as in Figure 1b. The solution to exclude these cases is to forbid the existence of such a fix-point.

► **Definition 2.** A synchronizing CA is *strongly synchronizing* if it is synchronizing and does not have a uniform fix-point (formally, there does not exist  $b \in Q$  such that  $F({}^\omega b^\omega) = {}^\omega b^\omega$ ).

Intuitively, synchronizing CA correspond to CA whose attractors are a finite set of cycles (a unique cycle for fully synchronizing). Strongly synchronizing CA are the specific case where none of these attractors is reduced to a single configuration.

Several results are known in slightly different contexts. In particular, M. Delacourt's construction in [4] can be used to construct a cellular automaton which converges toward a uniform cycle of length two for *almost all* configurations or for *almost all* periodic configurations [3]. In the case of probabilistic cellular automata, N. Fatès has shown a wide range of rules displaying full strong synchronisation [5].

## 2 On general and periodic configurations

We now show that, over the set of all configurations and over the set of periodic configurations, strong synchronisation is not possible. Moreover, in the general case, the notion of synchronisation corresponds exactly to the well-known notion of nilpotency (see [7] for more information about this notion).

► **Lemma 3.** *A cellular automaton  $(Q, f)$  is synchronizing over the set of all configurations if and only if it is nilpotent (i.e.,  $\exists a \in Q$  and  $N \in \mathbb{N}$  such that,  $\forall c \in Q^{\mathbb{Z}}$ ,  $F^N(c) = {}^\omega a^\omega$ ).*

**Proof.** Trivially, nilpotency implies (full) synchronisation and existence of a fix-point.

For the other direction, the basic idea is to consider a *universe* configuration  $c_\Omega \in Q^{\mathbb{Z}}$  containing all finite words of  $Q^*$  as factors. By definition, if the automaton is synchronizing, there exist  $n_\Omega \in \mathbb{N}$  and  $a \in Q$  such that  $F^{n_\Omega}(c_\Omega) = {}^\omega a^\omega$ . This implies that, for any word  $u \in Q^{2n_\Omega+1}$ , it holds  $f^{n_\Omega}(u) = a$ . Hence, for any configuration  $c \in Q^{\mathbb{Z}}$ , we have  $F^{n_\Omega}(c) = {}^\omega a^\omega$ . ◀

► **Lemma 4.** *Any synchronizing cellular automaton  $(Q, f)$  over the set of periodic configurations has a fix-point.*

**Proof.** Let us first consider the sequence of states  $(q_n)_{n \in \mathbb{N}}$  defined by  $q_0 = q_1 = q_2 = q \in Q$  and for any  $n \geq 2$ ,  $q_{n+1} = f(q_{n-2}, q_{n-1}, q_n)$ . Since  $Q$  is finite, there exists  $x, y > 2, x \neq y$  such that  $(q_{x-1}, q_x, q_{x+1}) = (q_{y-1}, q_y, q_{y+1})$ . Let us denote as  $\sigma : Q^{\mathbb{Z}} \rightarrow Q^{\mathbb{Z}}$  the *shift* defined by, for any  $c \in Q^{\mathbb{Z}}$  and  $p \in \mathbb{Z}$ ,  $\sigma(c)_p = c_{p+1}$ . By construction,  $F({}^\omega(q_x q_{x+1} \dots q_{y-1})^\omega) = \sigma^2({}^\omega(q_x q_{x+1} \dots q_{y-1})^\omega)$  and,  $\forall n \in \mathbb{N}$ ,  $F^n({}^\omega(q_x q_{x+1} \dots q_{y-1})^\omega) = \sigma^{2n}({}^\omega(q_x q_{x+1} \dots q_{y-1})^\omega)$ . Since the cellular automaton is synchronizing, there exists  $a \in Q$  such that  ${}^\omega(q_x q_{x+1} \dots q_{y-1})^\omega = {}^\omega a^\omega$  which implies  $F({}^\omega a^\omega) = \sigma^2({}^\omega a^\omega) = {}^\omega a^\omega$ . ◀

In fact, the previous proof can also prove the following: if a cellular automaton does not have a fix-point, then there exists a non-uniform periodic configuration on which it behaves as a power of the shift.

Using the uniform fix-point as a sink-hole, it is easy to construct a wide range of possible behaviours including non fully synchronizing cellular automata. For example, the automaton on  $Q = \{e, 0, 1\}$  with the rule  $f(0, 0, 0) = 1$ ,  $f(1, 1, 1) = 0$  and  $f$  outputs  $e$  for any other case gives a synchronizing CA on periodic configurations which either stays in the cycle  ${}^\omega 0^\omega, {}^\omega 1^\omega$  (starting from one of these configurations) or goes into  ${}^\omega e^\omega$  otherwise.

## 3 Our main result

At this point, our goal is to find a fully and strongly synchronizing cellular automaton. Indeed, we give in this section an example for the set of automata working over finite configurations of prime period.

► **Theorem 5.** *There exists a fully and strongly synchronizing cellular automaton over the set of periodic configurations of prime period (or unit period).*

The rest of the section is devoted to construct a cellular automaton  $(Q, f)$  for which any periodic configuration (of prime period) converges toward the length two cycle containing the two configurations  ${}^\omega 0^\omega$  and  ${}^\omega 1^\omega$  ( $0, 1 \in Q$ ).

The construction is done by using three layers of set of states  $Q = N \times K \times C$ . At first, Section 3.1 gives a non-deterministic automaton  $(N, f)$ , on partially synchronized growing intervals. This automaton is extended into a deterministic cellular automaton on set of

■ **Table 1** Set  $N$  of states. Symbol on top and left defines subset of the state set for convenience.

	$L$ (left border)	$I$ (interior)	$R$ (right border)	$D$ (dual)
$B$ (before)				
$S_r$ (right signal)				
$S_l$ (left signal)				
$Er$ (erasure)				
$A$ (after)				
$V$ (values)				

states  $N \times K \times C$  where  $K$  and  $C$  are layers controlling which of the possible rules of  $f$  the  $N$ -layer chooses from. In Section 3.2, we add the  $K$ -layer based on an aperiodic tiling to ensure that any configuration will contain only full intervals. In Section 3.3, we define the  $C$ -layer controlling disappearing behaviour of intervals, using (at last) primality, to achieve the result.

### 3.1 Intervals

The central element of our construction is the notion of interval that contains an increasing part of locally synchronized computation. In our cellular automaton, an interval is based on a portion of the configuration between a *left border* depicted by symbol  $\boxed{\leftarrow}$ , and a *right border*  $\boxed{\rightarrow}$ , containing *interior*. This interval is endowed with a *signal* going left  $S_l$  ( $\boxed{\diagdown}$ ,  $\boxed{\diagup}$  and  $\boxed{\diagup}$ ) and right  $S_r$  ( $\boxed{\diagup}$ ,  $\boxed{\diagdown}$  and  $\boxed{\diagdown}$ ). To ensure that there is exactly one signal in each interval, all cells in the interval, not containing the signal, indicate whether they are *before*  $\rightarrow$  or *after*  $\leftarrow$  the signal. The states resulting from the product of these two elements are detailed in Table 1 alongside the subsets of states used in the rest of the paper.

► **Definition 6** (Interval). An *interval* is a word of the form  $(LI^*R) \cap (B^*(Sl \cup Sr \cup Er)A^*)$  or an element of  $D$ .

The main idea of the construction is to divide the configuration into growing intervals that will “compete” with each other until only one is left. In this section, we firstly assert the behaviour of intervals on their own.

First and foremost, the transition rule is only defined on sequences of intervals separated by portion of values. When the neighbourhood is not locally valid, the rule outputs a new *initial interval*  $\boxed{\leftarrow}$ .

To achieve growth (Figure 2a), the signal inside the interval goes back and forth three times. The third time the signal reaches the left border, the interval grows of one to the left by moving its left border.

Moreover, on conditions which will be detailed in Section 3.3, the signal can decide to go into *erasure* mode (see Figure 2b) when bouncing on the right border. In this case, it goes back to the left border using the  $\boxed{\diagdown}$  signal and then erases the interval from left to right by pushing the left border with  $\boxed{\leftarrow}$  and filling the liberated space with  $\boxed{0}$  or  $\boxed{1}$ . This last event can also occur if a grow cannot be achieved due to an obstruction on the left on the interval (see Figure 2c).

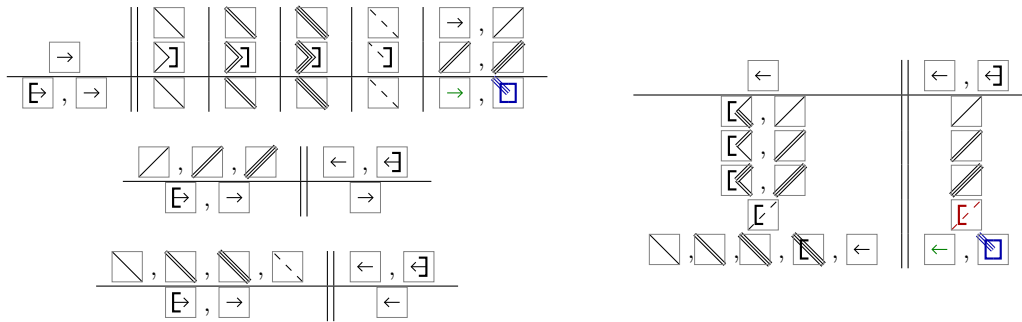
This behaviour is implemented with rules depicted in Table 2.

In this section, we will focus on the set of *pre-image*  $f^{-1} : N^n \rightarrow N^{n+2}$  of finite factors of intervals.

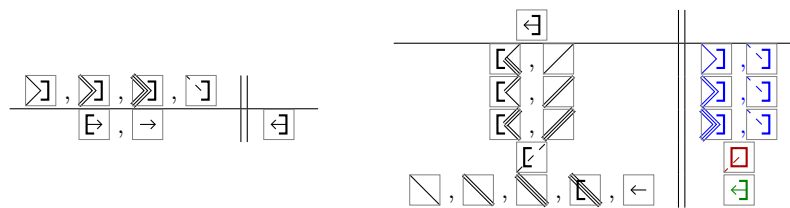
► **Lemma 7.** Any *pre-image* of an interval, except the initial one  $\boxed{\leftarrow}$ , contains an interval.



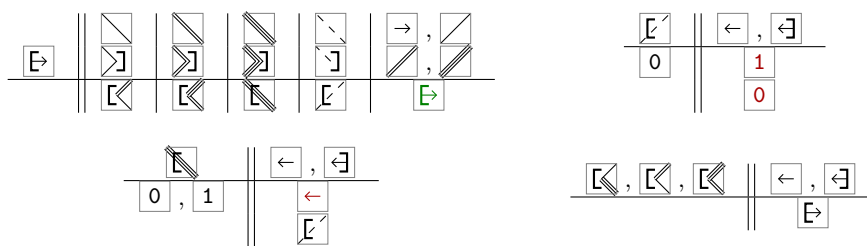
■ **Table 2** Local function of the cellular automaton  $(N, f)$ .



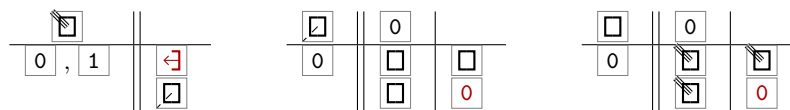
(a) When the central element is in  $I$



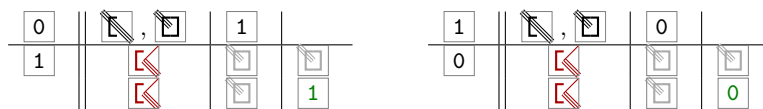
(b) When the central element is in  $R$



(c) When the central element is in  $L$



(d) When the central element is in  $D$

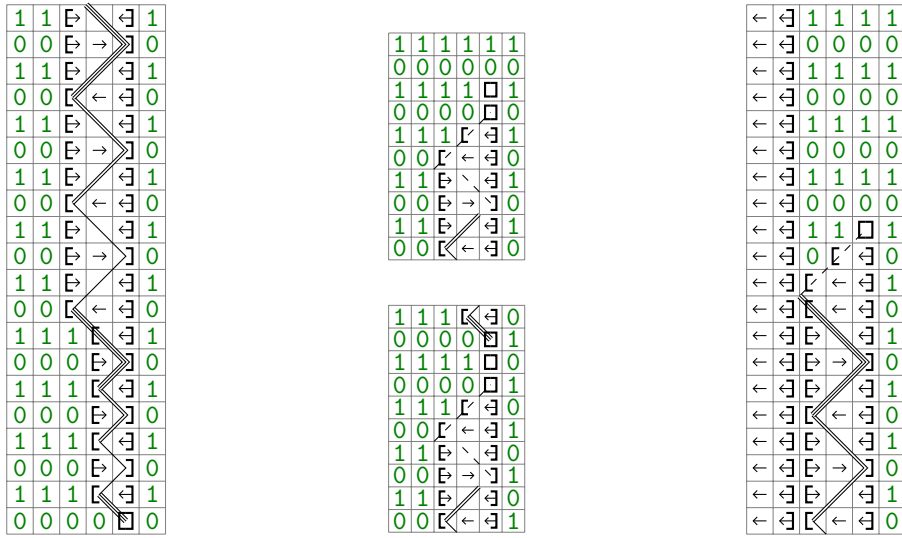


(e) When the central element is in  $V$

The local function is given on the form  $\frac{c}{l} \parallel \frac{r}{f(l, c, r)}$ .

Any undefined case outputs  $\square$ . When  $l$  or  $r$  is missing, it indicates all states locally valid that have not been defined elsewhere.

Green indicates when the central element is unchanged; red indicates when the status of the central element is changed. Blue indicates non-deterministic transitions which will be treated in Section 3.2 for dark blue and in Section 3.3 for light blue. Gray indicates cases which are in fact unused after the initial step due to Lemma 10.



(a) Growth of an interval      (b) Erasure of an interval      (c) left blocked

■ **Figure 2** Desired basic behaviour for interval and its implementation (note that this behaviour is not currently deterministic).

**Proof.** Let  $(c_i)_{1 \leq i \leq n} \in N^n$  be an interval and  $(b_i)_{0 \leq i \leq n+1} \in N^{n+2}$  be one element of its pre-image as depicted below. Cases where  $n = 1$  ( $c = \square$ ) or  $c = \square$ ) can be checked directly looking at the Table 2. For the other cases, we must have  $c_1 \in L$ ,  $c_n \in R$  and  $(c_i)_{2 \leq i \leq n-1} \in I$ .

	$c_1$	$c_2$	$\dots$	$c_{n-1}$	$c_n$	
$b_0$	$b_1$	$b_2$	$\dots$	$b_{n-1}$	$b_n$	$b_{n+1}$

Since  $f(x)_i \in R$  if and only if  $x_i \in R \cup D$  (see Table 2), then among  $b$ , only  $b_n \in R$ . Moreover,  $f(x)_i \in L$  implies  $x_{i-1}, x_i$  or  $x_{i+1} \in L$ . At last, since  $c$  does not contain the state  $\square$ , it must use the transitions explicitly defined in Figure 2 which force by construction the word between the two latter positions to be an interval. ◀

Now, let us focus on what happens in the pre-image of proper factors of intervals. The basic idea is that any proper factor must be issued from a “bigger” factor in the pre-image. To prove this, we first need to introduce a specific notion of size.

► **Definition 8 (Size).** Given a factor  $f \in N^*$  of an interval, the *size*  $s(f)$  corresponds to the number of symbols except counting only one half for symbol  $\square$ .

For example:  $s(\square) = 1$ ,  $s(\square \square) = 2$ ,  $s(\square \leftarrow \leftarrow) = 2.5$  and  $s(\leftarrow \square \leftarrow) = 3$ .

► **Lemma 9.** Any pre-image of a proper factor of an interval contains a factor of strictly greater size.

**Proof.** The first remark is that the pre-image can indeed be an interval. The proof is done by refining the previous proof. Let us consider the case when the proper factor is a prefix of an interval (the suffix case is similar). This implies that  $c_n \in I$ . Looking at the transition table, we have that  $b_n$  must be in  $I$  and  $b_{n+1}$  is either in  $I$  or in  $R$ . Then we have  $s(b) > s(c)$ , unless the left border in  $b$  is  $b_2$ ; but this is only possible when  $c_1 = \square$ , and then we have  $s(b) = n$  and  $s(c) = n - 1 + \frac{1}{2} = n - \frac{1}{2}$ . ◀

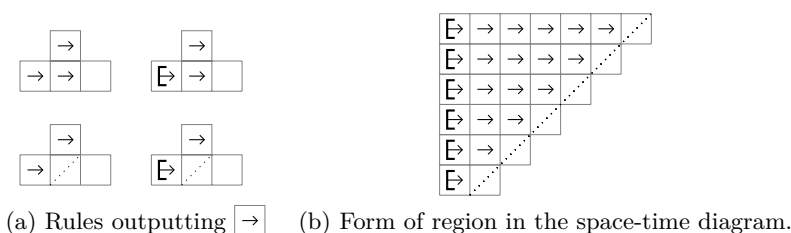
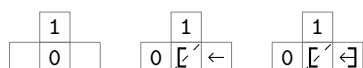


Figure 3 Regions of  $\rightarrow$  (here,  $\diagdown$  denotes either  $\diagup$ ,  $\diagdown$  or  $\diagup$ .)

At last, one can easily see that some words do not appear in any image:

► **Lemma 10.**  $10$  and  $01$  are not possible in the image.

**Proof.** Looking at Table 2, we can see that the only way to output a  $1$  are the cases:



Since outputting a  $0$  requires that there is no  $0$  in the neighbourhood and that neither  $\leftarrow$  nor  $\leftarrow$  can generate  $0$ , no  $0$  can occur next to a  $1$ . ◀

With these first non-deterministic rules, any periodic configuration can be divided into intervals or short-lived proper factors of them (since any letter is itself a proper factor) separated by uniform spaces of 0 or 1. The only exception is when the configuration contains only one periodic proper factor of an interval, which can only be of the form  $\omega \rightarrow \omega$  or  $\omega \leftarrow \omega$ . As we want to keep only full intervals, the option selected is to add one layer to ensure the two following properties: infinite configurations consisting only of interior will eventually disappear ; proper factors of intervals cannot be created. This is done in the following section.

### 3.2 Ensuring intervals with an aperiodic tiling

To ensure the presence of at least one interval, we want to detect inside uniform  $\rightarrow$  or  $\leftarrow$  regions the periodicity without perturbing the normal behaviour of such regions. To do this, we shall add a layer containing an aperiodic tiling over each region independently. In this section, we shall only detail the case  $\rightarrow$  since  $\leftarrow$  is similar.

The first remark is that state  $\rightarrow$  can only be generated by the rules depicted in Figure 3a leading to a region in a bottom triangular form (see Figure 3b).

Then, we want to fill this region with a south-west deterministic tiling. Formally, a Wang tile  $t$  is a square tile with coloured edges, as represented in Figure 4a. It is given by a quadruplet  $(t_e, t_w, t_n, t_s)$  of symbols, called colours. A tile-set  $\tau$  is a finite set of Wang tiles. A tiling of the plane by  $\tau$  is a map  $t$  from the discrete plane  $\mathbb{Z}^2$  to  $\tau$  such that two tiles that share a common edge agree on its colour: For all integers  $i, j$  we have  $t(i, j)_e = t(i + 1, j)_w$  and  $t(i, j)_n = t(i, j + 1)_s$ . A tile set is said to be south-west deterministic if there is at most one possible tile for every possible  $(t_s, t_w)$  (see Figure 4b). A tile-set  $\tau$  is aperiodic if there exists at least a valid tiling of the plane by  $\tau$  but no periodic tiling.

In this paper, we shall use Kari’s tile-set [7] (or more precisely, its horizontal flip). This tile-set is based on Robinson’s aperiodic tiling [9] and was introduced to study nilpotency on cellular automata. It can be described with two layers (corresponding to a Cartesian product). The first one is just a regular 4 coloured grid. The second one is usually depicted using lines. The tiles are depicted in Figure 5.

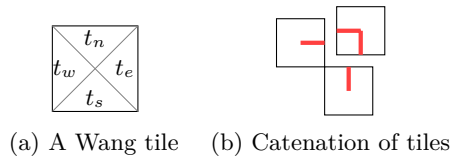
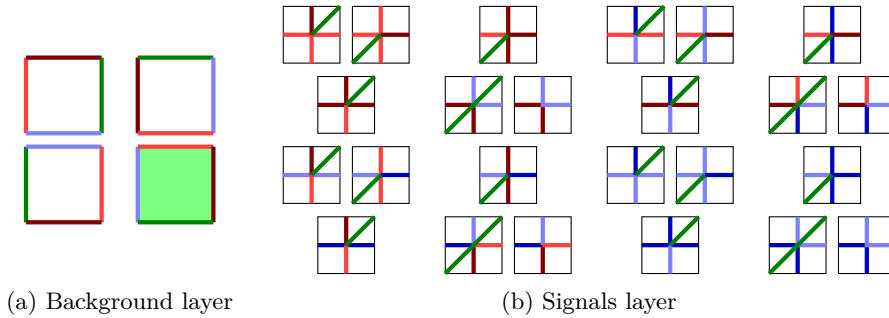


Figure 4 Wang tiles.



Only signal tiles whose both dark left and bottom lines ( , , , , , , , ) can appear above the background .

Figure 5 A simplified representation of the south-west deterministic aperiodic tile-set.

► **Theorem 11** (J. Kari, 1992 [7]). *The tile set in Figure 5 is aperiodic and south-west deterministic.*

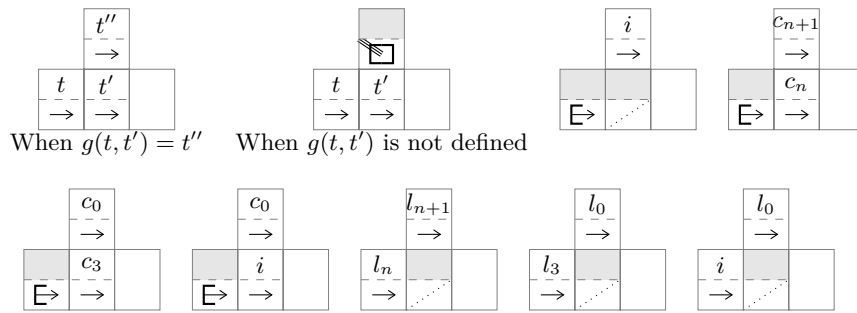
This tiling can locally be implemented as a one-way cellular automaton  $g$  over  $\tau$  by associating, for  $t, t' \in \tau$ , the only tile  $g(t, t')$  fitting (by south-west determinism). One can thus use this to construct the layer in the inside of the region by enhancing the rules as depicted in Figure 6. In case no such tile exists, the transition is defined as . It can be noted that the additional rule is compatible with Table 2 and even more, it suppresses both cases of non-determinism in case (a).

The previous idea is only valid for the inside and it leaves the case of borders open. For those, we need another property which is the existence of a “regular” north-east quarter of the plane. This is the case for the previous tiling, in particular we shall use the partial tiling depicted in Figure 7. Due to the self-similarity of the tiling, one can see the highlighted column (resp. line) consists of a periodic sequence of size 4  $\{c_0, c_1, c_2, c_3\}$  (resp.  $\{l_0, l_1, l_2, l_3\}$ ) with the possible exception of the common initial tile  $i$ . We can use this property to define the last remaining cases of Figure 3a to ensure the filling of the region exists and is as depicted in Figure 6.

Let us now prove that the additional layer does indeed do what we want: forbid uniform configuration of or , and forbid proper prefix of intervals. This is done in the two following lemmas.

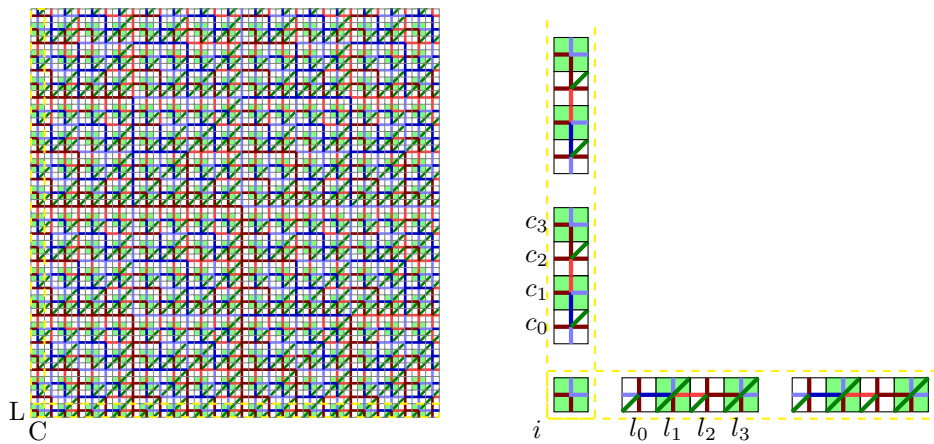
► **Lemma 12.** *On a periodic configuration of period  $N$ , after less than  $|\tau|^N$  steps of the automaton, the projected configurations  $\omega \rightarrow^\omega$  and  $\omega \leftarrow^\omega$  cannot appear.*

**Proof.** By construction, such configurations contain a layer with the aperiodic tiling for every pre-image. By contradiction, if the configuration appears after  $|\tau|^N$  steps, we can extract  $|\tau|^N$  configurations for the tiling layer. In this case, at least two of them are identical and can be glued into a valid periodic tiling leading to a contradiction. ◀



Tiles  $i$ ,  $c_i$  and  $l_i$  are the specific tiles on the border in Figure 7. Tile  $i$  is the bottom-left corner. Sequence  $c_0, c_1, c_2, c_3$  (resp.  $l_0, l_1, l_2, l_3$ ) is the periodic sequence in the leftmost column (resp. bottom line).

■ **Figure 6** Additional aperiodic layer.



■ **Figure 7** Valid quarter of the plan tiling with regular diagonals.

Let us call  $T_{\max}^K(n)$  the bound in the previous lemma. This bound (which is independent of the tiling) is exponential in  $N$ . For the specific tiling used here, a better study could almost surely prove a quadratic bound.

More than just eliminating uniform configurations we do not want, this layer avoids the creation of proper prefix of intervals.

► **Lemma 13.** *On a periodic configuration of period  $N$ , after at most  $2N + \max(T_{\max}^K(N), 2N)$  steps, the configuration does not contain any proper factor of interval.*

**Proof.** By contradiction, assume there exists such a proper factor. We can look at its sequence of pre-images.

If this sequence does not contain a full interval, by Lemma 9, it must increase in size (and thus in length) and thus reaches one uniform configuration  $\omega \xrightarrow{\rightarrow} \omega$  or  $\omega \xleftarrow{\leftarrow} \omega$  in less than  $2N$  steps which contradicts Lemma 12.

Let us consider now the step when the pre-image  $b$  is an interval and the image  $c$  is a strict factor. The key point is that this case can only happen when there is an error in the aperiodic layer outputting a symbol  $\boxed{\rightarrow}$ , otherwise, the image of an interval is an interval. Without loss of generality, we can assume that this error occurs over the symbol  $\boxed{\rightarrow}$ .

In all cases, the error over  $\boxed{\rightarrow}$  occurs in a triangle of such states. However, the transitions are made so that any point of the triangle behaves as the tiling depicted in Figure 7.

Thus there cannot be errors in the tiling. This implies that  $c$  is an interval, which is a contradiction. ◀


Since no proper prefix of interval may be created after a certain time denoted as  $T_R(N)$ , we can thus give a precise characterization of configurations which can appear after some initial transient behaviour.

► **Proposition 14.** *After  $T_R(N)$  steps, the configuration consists only of intervals possibly separated by uniform portions of 0 or 1.*

**Proof.** This result follows directly from the two previous Lemmas and Lemma 10. ◀











Moreover, in this case, we have some additional very useful property.

► **Lemma 15.** *After  $T_R(N)$  steps, no right border is created and the number of intervals in the configuration is decreasing.*

**Proof.** It is sufficient to remark, looking at Figure 2, that only transitions outputting  can create a right border and that it cannot happen in the conditions depicted by the previous lemma. Either because of the form of configurations (for the gray cases) or because there cannot be an error (by the proof of Lemma 13). ◀

### 3.3 Comparing intervals

The last point of the construction is to effectively synchronize. The idea is to keep the largest interval by comparing any two adjacent intervals and suppressing the smaller one. Here, we shall at last use the property of primality which will ensure, in some sense, that such a largest interval exists.

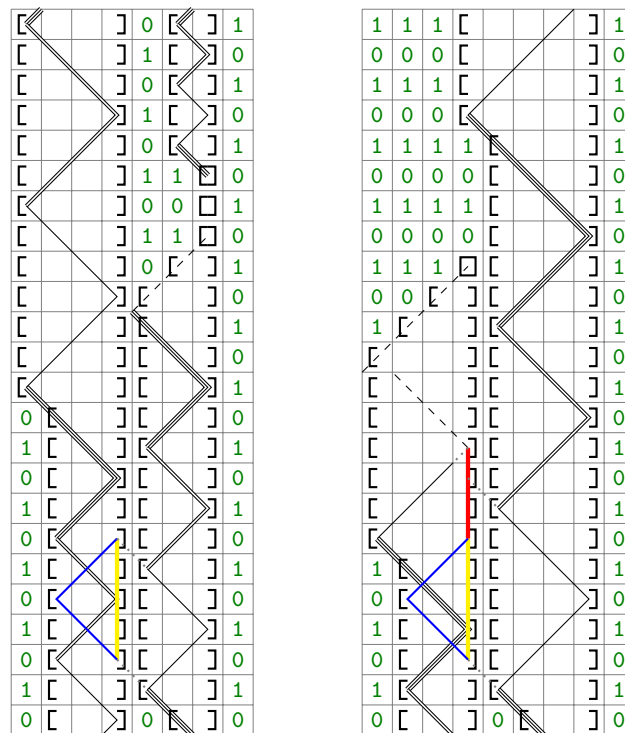
For this last proof, we add a layer  $C$  which contains a new full speed signal (, ) and two vertical markers (, ). This layer works in the following way (see Figure 8 for a global view): Each time it encounters a new right neighbour (that is, when an interval on its left has grown and reaches it), a comparison is started. This comparison launches the new signal inside the left interval and adds a static marker to indicate a undergoing comparison. The new signal makes a round trip around the left interval whereas the signal (, ) makes a similar trip in the right interval. The result of the comparison is determined according to which of these signals comes back first on the marker. If the new signal arrives first (meaning the left interval is strictly smaller than the right one), then a mark is left to call for the destruction of the left interval by ensuring that the next time a signal ,  or  arriving is transformed into  (lifting the last non-deterministic case). In the other case, nothing is done since the right interval will erase itself if it cannot grow.

One notable property is that the signal takes some time to go back and forth so the comparison is not immediate and synchronous.

Let us now prove the constructed cellular automaton achieves the desired synchronizing behaviour. For a fixed period  $N$ , the number of possible configurations is finite and thus any configuration eventually reaches a cycle. In this cycle, the number of intervals is constant. Let us define as  $s_\infty$  the maximal size of an interval inside this cycle.

► **Lemma 16.** *The case  $s_\infty = 1$  is not possible.*

**Proof.** By contradiction assume that  $s_\infty = 1$ . Since all intervals are of maximal size 1, there is no  $L$ ,  $R$  or  $I$  states in the configuration. Thus, it only consists of states either in  $V$  or in  $D$ . After some time, those are stable since no interval is created nor destroyed. Let us look



(a) left neighbour is larger (b) left neighbour is strictly smaller

■ **Figure 8** Comparison of intervals.

at some state in  $D$  at this point. Looking at the transition rule, the only possibility for this state to stay in  $D$  is to alternate the sequence  $\square, \square, \square$  over time. However, the transition  $\square \rightarrow \square$  requires its left state to be in  $D$  whereas the one  $\square \rightarrow \square$  requires it to be in  $V$ , which is not possible. ◀

► **Lemma 17.** *The case  $s_\infty \geq 2$  is not possible.*

**Proof.** By contradiction assume that  $s_\infty \geq 2$ . There exists a configuration with an interval of size  $s_\infty$ . This interval must be persistent over time and thus makes a cycle going from  $\square$  up to an interval of size  $s_\infty$  then erases itself to  $\square, \square$  and then goes back to  $\square$ . For the proof, let us look at the moment the interval goes into erasing mode. This can be due to two different cases: either because its growth was denied or because a comparison has produced an erasing signal (see Figure 2). Let us review the two cases.

**When the interval disappears because of left blocking.** In this case, the first trivial remark is that there must be an interval to the left. As right borders are persistent in this case conditions, there must be an interval directly to the left. Let us now look at the moment  $t_0$  our interval has grown to size  $s_\infty$ . It has thus launched a comparison.

If the interval to the left is of size  $s < s_\infty$ , then the comparison would have detected this before instant  $t_0 + 2(s - 1)$  which is the time for the signal to do a round trip. Moreover, this comparison would have resulted, for the left neighbour, in the appearance of an erasure signal after at most  $2(s - 1)$  steps. At last, this results in the erasure of the left interval in less than  $2(s - 1) + 2$  steps. Since our interval is alive for at least  $6 * (s_\infty - 1)$  steps, the left

interval would have disappeared before our interval even tries to grow which contradicts it being blocked.

Thus we can assume that the interval to the left is of size  $s_\infty$  when comparing. It implies that it will also erase itself later which can only be for the same reason (blocked) as the initial interval.

By iterating the proof, we can see that there is an infinite sequence of intervals which are all of size  $s_\infty$ . Even if this sequence is found at different times, since right borders do not move, appear, or disappear, this means that the configuration has a right border every  $s_\infty$  cells. At last, since the period is a prime number, this implies that there is only one interval (of size  $s_\infty$ ). However, in this case, it is easy to see that it goes into the uniform configuration  $\omega \boxed{0}^\omega$  at the end of the erasure, which contradicts  $s_\infty > 0$ .

**When the interval disappears because it is erased.** This proof is very similar to the previous one. This case is only possible if the interval is strictly smaller than its right neighbour. This can indeed happen if the comparison was done just before an increase. In this case, the right neighbour was of size  $s_\infty$  (before). Moreover, since the erasure is, as in the previous case, done before the right neighbour grows, it implies that the latter also disappears because of erasure. Thus, we are in the same case, when the configuration is split into intervals regularly spaced which can only mean there is only one interval. This leads to the same contradiction as before. ◀

#### 4 Generalisation and extension

To summarise, this paper shows that synchronisation cannot be achieved in a spatially uniform deterministic context with only local information. However, it gives one sufficient and surprising global additional condition (primality) that can be exploited to achieve synchronisation. Moreover, an anonymous referee has hinted that the proof can be extended to the odd case by forcing the lifetime of an interval to be even. In that case, the contradiction for Lemma 17 is achieved by the fact that at least two consecutive intervals should have a even shift between which would results in their disappearance.

The construction done in the previous section can be extended to achieve one cycle of any size. Once this is done, the same remark as in the end of Section 2 applies: using Cartesian product, it is easy to add any additional distinct number of cycles of any length.

Nevertheless, our construction is quite complex and relies heavily on the fact that the dimension is one. This opens immediately the question of the validity of this result in higher dimension; especially since the extension of the primality condition is not clear. One other extension is to consider robust computation by introducing in the transition rule a small probability of error (see [6]). In this case, it may even be possible to get rid of the primality requirement.

---

#### References

- 1 Nicolas Bacquey. Complexity classes on spatially periodic cellular automata. In Ernst W. Mayr and Natacha Portier, editors, *STACS*, volume 25 of *LIPICs*, pages 112–124. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPICs.STACS.2014.112.
- 2 Karel Čulík II. Variations of the firing squad problem and applications. *Information Processing Letters*, 30(3):152–157, 1989.



- 3 Martin Delacourt. *Automates cellulaires : dynamique directionnelle et asymptotique typique*. PhD thesis, université de Provence, 2011.
- 4 Martin Delacourt. Rice's theorem for  $\mu$ -limit sets of cellular automata. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *Automata, Languages and Programming – 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II*, volume 6756 of *Lecture Notes in Computer Science*, pages 89–100. Springer, 2011. doi:10.1007/978-3-642-22012-8\_6.
- 5 Nazim Fatès. Remarks on the cellular automaton global synchronisation problem. In Jarkko Kari, editor, *Cellular Automata and Discrete Complex Systems – 21st IFIP WG 1.5 International Workshop, AUTOMATA 2015, Turku, Finland, June 8-10, 2015. Proceedings*, volume 9099 of *Lecture Notes in Computer Science*, pages 113–126. Springer, 2015.
- 6 Péter Gács. Reliable cellular automata with self-organization. *Journal of Statistical Physics*, 103(1-2):45–267, 2001. doi:10.1023/A:1004823720305.
- 7 Jarkko Kari. The nilpotency problem of one-dimensional cellular automata. *SIAM Journal on Computing*, 21(3):571–586, 1992. doi:10.1137/0221036.
- 8 Jacques Mazoyer. On optimal solutions to the firing squad synchronization problem. *Theoretical Computer Science*, 168(2):367–404, 1996. doi:10.1016/S0304-3975(96)00084-9.
- 9 Raphael Robinson. Undecidability and nonperiodicity for tilings of the plane. *Inventiones Mathematicae*, 12, 1971. doi:10.1007/BF01418780.
- 10 John von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966. doi:10.1002/asi.5090180413.
- 11 Stephen Wolfram. *A new kind of science*. Wolfram Media Inc., Champaign, Illinois, United States, 2002.



# Word Equations Where a Power Equals a Product of Powers

Aleksi Saarela

Department of Mathematics and Statistics, University of Turku, Turku, Finland  
amsaar@utu.fi

---

## Abstract

We solve a long-standing open problem on word equations by proving that if the words  $x_0, \dots, x_n$  satisfy the equation  $x_0^k = x_1^k \cdots x_n^k$  for three positive values of  $k$ , then the words commute. One of our methods is to assign numerical values for the letters, and then study the sums of the letters of words and their prefixes. We also give a geometric interpretation of our methods.

**1998 ACM Subject Classification** G.2.1 Combinatorics, F.4.3 Formal Languages

**Keywords and phrases** Combinatorics on words, Word equations

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.55

## 1 Introduction

We say that words  $x_0, \dots, x_n$  *commute* if  $x_i x_j = x_j x_i$  for all  $i, j \in \{0, \dots, n\}$ . One of the early results on word equations is the result of Lyndon and Schützenberger [13] that if  $x^k = y^m z^n$  for some words  $x, y, z$  and numbers  $k, m, n \geq 2$ , then  $x, y, z$  commute (or, equivalently,  $x, y, z$  are powers of a common word). Many generalizations have been studied, for example by Lentin [11] and by Shyr and Yu [17]. We are interested in the generalizations where the right-hand side can have more than two powers, but all exponents are equal (here  $k \geq 1$ ):

$$x_0^k = x_1^k \cdots x_n^k. \quad (1)$$

In particular, we are interested in systems of equations where the words  $x_0, \dots, x_n$  satisfy (1) for many values of  $k$ . An even more general family of equations that could be studied is formed by equations of the form

$$s_0 x_1^k s_1 \cdots x_m^k s_m = t_0 y_1^k t_1 \cdots y_n^k t_n. \quad (2)$$

Let us briefly mention some connections and applications of the above equations (1) and (2) (more details can be found in the references given). The first application is in the theory of test sets. A subset  $K$  of a language  $L$  is called a test set if, for all morphisms  $f$  and  $g$ , either  $f(x) \neq g(x)$  for some  $x \in K$  or  $f(x) = g(x)$  for all  $x \in L$ . This means that to check whether  $f$  and  $g$  agree on  $L$ , it is sufficient to check whether they agree on  $K$ . Connections between the above equations and test sets are explained, for example, in [7]. As a second application, the equations come up when studying pumping properties of formal languages. For example, in the article [3], pumping the computations of transducers in two places leads to equations of the form (2) with  $m = n = 2$ . The equations naturally arise in many other settings as well. As a third application, equations (1) are related to the construction of large independent systems of word equations [9, 15]. In fact, a counterexample to Conjecture 1 (stated below and proved in this article) would have improved the best known lower bounds for the size of independent systems. Our last example is that the pair of equations (1) for



© Aleksi Saarela;

licensed under Creative Commons License CC-BY

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 55; pp. 55:1–55:9

Leibniz International Proceedings in Informatics



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

$k = 1, 2$  is connected to Sturmian words, and this connection leads to a large family of interesting solutions [14].

The main question about equations (1) is when do they imply that  $x_0, \dots, x_n$  commute. Sometimes a single equation is enough: Appel and Djorup [1] proved that if  $k = n$  in (1), then the words  $x_0, \dots, x_n$  must commute. Their result was generalized by Harju and Nowotka [5] for certain equations which have many different exponents  $k_1, k_2, \dots$  instead of just one exponent  $k$ . On the other hand, there are many examples of words  $x_0, \dots, x_n$  such that  $x_i x_j \neq x_j x_i$  for some  $i, j$ , but (1) holds for two different values of  $k$ . For instance,  $(ababa)^k = (ab)^k a^k (ba)^k$  for  $k \in \{1, 2\}$ . No such examples are known for three different values of  $k$ . In fact, the following conjecture is well-known.

► **Conjecture 1.** *If  $x_0, \dots, x_n$  are words and  $k_1, k_2, k_3 \geq 1$  are different numbers such that (1) holds for  $k \in \{k_1, k_2, k_3\}$ , then  $x_0, \dots, x_n$  commute.*

In some form, this conjecture has been open for at least about two decades. The case  $\{k_1, k_2, k_3\} = \{1, 2, 3\}$  appeared as a question (and was proved for  $n \leq 5$ ) in an article by Hakala and Kortelainen [4], and as an explicit conjecture in an article by Plandowski [15], and a prize for a proof was offered by Holub in 2009<sup>1</sup>. The case where one of  $k_1, k_2, k_3$  is 1 was asked as a question in [6], and the case  $k_1, k_2, k_3 \geq 2$  was proved in [7] by Holub. If (1) holds for  $k = 1$ , then the other equations can be replaced by

$$(x_1 \cdots x_n)^k = x_1^k \cdots x_n^k, \quad (3)$$

so the conjecture often appears in the following form: If (3) holds for two values of  $k \geq 2$ , then  $x_1, \dots, x_n$  commute.

For equations (2), we could ask the following question: For how many values of  $k$  does (2) need to hold to guarantee that it holds for all  $k \geq 0$ ? This was first studied by Kortelainen [10]. Currently it is known that  $m + n$  different values of  $k$  are sufficient [16]. Holub and Kortelainen [8] proved that a constant number of different values is sufficient in some special cases, but it is not known whether this is true in general.

In this article, we will prove Conjecture 1. As mentioned above, it would be sufficient to prove it in the case  $k_1 = 1$ , but our proof works for all values. This also makes the paper self-contained; to understand the proof, it is only necessary to be familiar with some basics of combinatorics on words, see, e.g., [12]. The possibility of applying the techniques of this paper to the more general equations (2) remains open.

The basic idea behind our proof is to assign numerical values for the letters in a specific way (so that the sum of the letters of  $x_0$  is zero), and then study the sums of the letters of words and their prefixes. We will also give a geometric interpretation of our methods; using geometric intuition was crucial when trying to prove the result.

## 2 Preliminaries

Let  $\Gamma$  be an alphabet. We can assume that  $\Gamma$  is a subset of  $\mathbb{R}$ . This allows us to define  $\Sigma(w)$  to be the sum of the letters of a word  $w \in \Gamma^*$ , that is, if  $w = a_1 \cdots a_n$  and  $a_1, \dots, a_n \in \Gamma$ , then  $\Sigma(w) = a_1 + \cdots + a_n$ . The mapping  $\Sigma$  is a morphism from the free monoid  $\Gamma^*$  to the additive monoid  $\mathbb{R}$ . Words  $w$  such that  $\Sigma(w) = 0$  are called *zero-sum words*.

<sup>1</sup> <http://www.karlin.mff.cuni.cz/~holub/soubory/prizeproblem.pdf>

The notation  $a_1 \cdots a_n$  of course means the word consisting of the letters  $a_1, \dots, a_n$  and not a product of numbers. When we actually want to compute the product of two numbers, it should be clear from context. If  $w_1, \dots, w_n$  are words, we can also use the notation

$$\prod_{i=1}^n w_i = w_1 \cdots w_n$$

for their concatenation.

Whenever the symbol  $\Gamma$  appears in this article, it is always used to denote an alphabet. Occasionally we will use other alphabets as well. All of them can be assumed to be subsets of  $\mathbb{R}$ . Alphabets are also assumed to be finite, unless otherwise specified.

Let  $a_1, \dots, a_k \in \Gamma$ . The *prefix sum word* of  $w = a_1 \cdots a_k$  is the word  $\text{psw}(w) = b_1 \cdots b_k$ , where  $b_i = \Sigma(a_1 \cdots a_i)$  for all  $i$ . Of course,  $\text{psw}(w)$  is usually not a word over  $\Gamma$ , but over some other alphabet. The word  $\text{psw}(w)$  has the same length as  $w$  and the last letter is  $\Sigma(w)$ .

The mapping  $\text{psw}$  is injective. It is not a morphism, but we can give a simple formula for the prefix sum word of a product by using the notation  $\text{psw}_r(w) = c_1 \cdots c_k$ , where  $r \in \mathbb{R}$  and  $c_i = b_i + r$  for all  $i$ . Then, for  $w_1, \dots, w_n \in \Gamma^*$ ,

$$\text{psw}(w_1 \cdots w_n) = \prod_{i=1}^n \text{psw}_{\Sigma(w_1 \cdots w_{i-1})}(w_i).$$

If  $w_1, \dots, w_n$  are zero-sum, then we have the simpler formula

$$\text{psw}(w_1 \cdots w_n) = \prod_{i=1}^n \text{psw}(w_i),$$

so in this case the mapping  $\text{psw}$  actually does behave like a morphism. For the  $n$ th power of a word  $w$ , we get the formula

$$\text{psw}(w^n) = \prod_{i=1}^n \text{psw}_{(i-1)\Sigma(w)}(w).$$

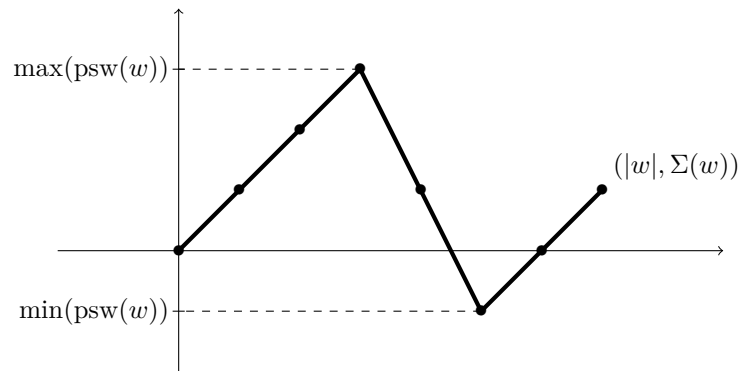
If  $w$  is zero-sum, then we have  $\text{psw}(w^n) = \text{psw}(w)^n$ .

Because letters are real numbers, there is a natural order relation for them. The largest and smallest letters in a word  $w$  can be denoted by  $\max(w)$  and  $\min(w)$ , respectively. The length of  $w$  is denoted by  $|w|$ , and the number of occurrences of a letter  $a$  in  $w$  is denoted by  $|w|_a$ . The size of a set  $S$  is denoted by  $|S|$ .

► **Example 2.** Let  $w = bbcaac$ , where  $a = 1$ ,  $b = 2$ , and  $c = -3$ . We have  $|w| = 6$ ,  $\max(w) = 2$ , and  $\min(w) = -3$ . Because  $\Sigma(w) = 2 + 2 - 3 + 1 + 1 - 3 = 0$ ,  $w$  is a zero-sum word. The prefix sum word of  $w$  is  $\text{psw}(w) = 241230$ , and  $\max(\text{psw}(w)) = 4$  and  $\min(\text{psw}(w)) = 0$ .

When studying words from a combinatorial point of view, the choice of the alphabet is arbitrary (except for the size of the alphabet). Therefore, we can assign numerical values to the letters in any way we like, as long as no two letters get the same value. The next lemma shows in a formal way that, given any word  $w$ , the alphabet can be normalized so that  $w$  becomes a zero-sum word.

► **Lemma 3.** *Let  $w \in \Gamma^*$ . There exists an alphabet  $\Delta$  and an isomorphism  $h : \Gamma^* \rightarrow \Delta^*$  such that  $h(w)$  is zero-sum.*



■ **Figure 1** Geometric representation of the word  $\text{psw}(w)$ , where  $w = aaabbaa$ ,  $a = 1$ , and  $b = -2$ . We have  $|w| = 7$ ,  $\Sigma(w) = 1$ ,  $\max(\text{psw}(w)) = 3$ , and  $\min(\text{psw}(w)) = -1$ .

**Proof.** If  $w$  is the empty word, it is already zero-sum. Otherwise, let  $d = \Sigma(w)/|w|$ . We can define an alphabet  $\Delta = \{a - d \mid a \in \Gamma\}$  and a morphism  $h : \Gamma^* \rightarrow \Delta^*$  by  $h(a) = a - d$  for all  $a \in \Gamma$ . Clearly,  $h$  is a bijection and therefore an isomorphism, and  $\Sigma(h(w)) = \Sigma(w) - d|w| = 0$ . ◀

By the next lemma, every zero-sum word can be written as a product of minimal zero-sum words in a unique way. Later we will use this to “compress” zero-sum words by replacing these factors by letters. The free monoid in the lemma can be either trivial (just the empty word) or infinitely generated.

► **Lemma 4.** *The set of zero-sum words over  $\Gamma$  is a free monoid.*

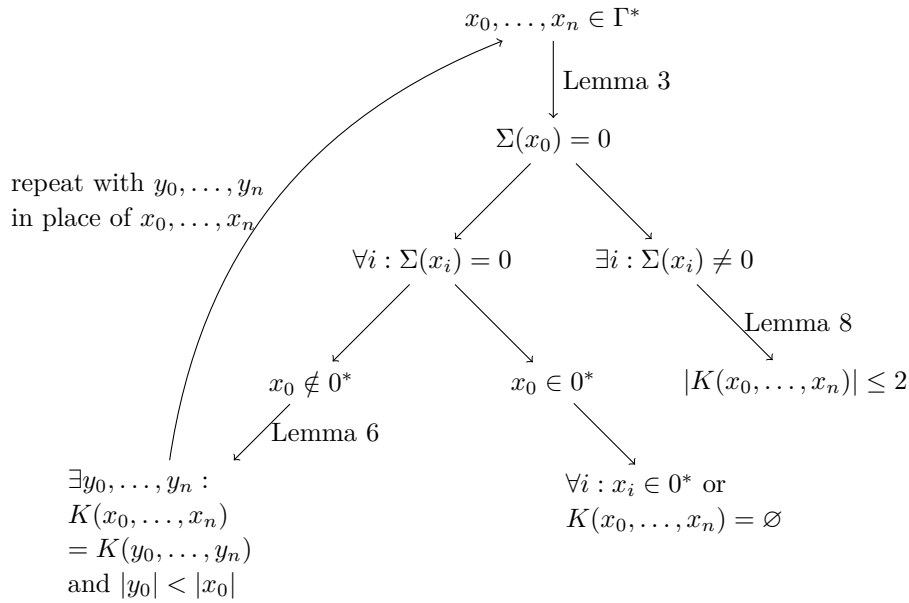
**Proof.** Clearly zero-sum words form a monoid. This monoid is right unitary, that is, if  $u$  and  $uv$  are zero-sum, then so is  $v$ . It is well-known that a right unitary submonoid of a free monoid is free. (The claim could easily be proved directly as well.) ◀

### 3 Geometric intuition

In this section, we give some geometric intuition, which is not necessary for the proofs, but it might be helpful in understanding them (at least it was helpful in inventing the proofs).

The above definitions have the following geometric interpretation: Let  $w = a_1 \cdots a_k$ . The word  $\text{psw}(w)$  (or the word  $w$  depending on the point of view) can be represented by a polygonal chain by starting at the origin, moving  $a_1$  steps up and one step to the right,  $a_2$  steps up and one step to the right, and so on. If  $\text{psw}(w) = b_1 \cdots b_k$ , then this curve is also obtained by connecting the points  $(0, 0), (1, b_1), \dots, (k, b_k)$ . The last point is  $(|w|, \Sigma(w))$ . If we start counting from the point  $(1, b_1)$  instead of  $(0, 0)$ , then the biggest  $y$ -coordinate is  $\max(\text{psw}(w))$  and the smallest  $y$ -coordinate is  $\min(\text{psw}(w))$ . See Figure 1 for an example. The word  $\text{psw}_r(w)$  could be represented in a similar way by starting at the point  $(0, r)$  instead of  $(0, 0)$ . The curve of  $\text{psw}(uv)$  consists of the curve of  $\text{psw}(u)$  followed by the curve of  $\text{psw}(v)$  translated in such a way that its starting point matches the endpoint of the curve of  $\text{psw}(u)$ .

The geometric interpretation is similar to the relation between Dyck words and Dyck paths, or the definition of Sturmian words as mechanical words. Representations of words as paths (or paths as words) can also be used in discrete geometry. This can, for example, lead to connections between word equations and tilings of a plane, see [2] for a survey.



■ **Figure 2** Structure of the proof.

**4** Idea of the proof

For  $x_0, \dots, x_n \in \Gamma^*$ , let

$$K(x_0, \dots, x_n) = \{k \in \mathbb{Z}_+ \mid x_0^k = x_1^k \cdots x_n^k\}.$$

We are going to prove that either  $x_0, \dots, x_n$  commute or  $K(x_0, \dots, x_n)$  contains at most two numbers. This proves Conjecture 1.

Before the formal treatment in Section 5, let us outline the strategy (also illustrated in Figure 2): First we use Lemma 3 to change the alphabet so that  $x_0$  becomes zero-sum. If all  $x_i$  are zero-sum, then either  $x_0$  is unary or we can use Lemma 6 to compress them so that the set  $K(x_0, \dots, x_n)$  is preserved. The compression can possibly be repeated several times, but only finitely many times. After that we end up either in the unary case, in which case also the original words  $x_i$  commute, or in the case where some  $x_i$  is not zero-sum. If some  $x_i$  is not zero-sum, then we can use Lemma 8 to prove that  $K(x_0, \dots, x_n)$  contains at most two numbers; this is the most complicated part of the proof. The idea in Lemma 8 is to compare the number of occurrences of a certain letter in the words  $\text{psw}(x_0^k)$  and  $\text{psw}(x_1^k \cdots x_n^k)$ . If these are different, then  $k \notin K(x_0, \dots, x_n)$ , because  $k \in K(x_0, \dots, x_n)$  is equivalent to  $\text{psw}(x_0^k) = \text{psw}(x_1^k \cdots x_n^k)$ .

► **Example 5.** Consider the case  $x_0 = abbaabbaab$ ,  $x_1 = abba$ ,  $x_2 = ab$ , and  $x_3 = baab$ .

Our alphabet is  $\{a, b\}$ , and we can choose arbitrary numerical values for  $a$  and  $b$ . We want  $x_0$  to be zero-sum, so let  $a = 1$  and  $b = -1$ . Then also  $x_1, x_2$  and  $x_3$  are zero-sum. We can do the compression, formalized in Lemma 6, by writing the words as products of the zero-sum words  $ab$  and  $ba$ , and then replacing  $ab$  with a letter  $c$  and  $ba$  with a letter  $d$ . We get the words  $y_0 = cdcdc$ ,  $y_1 = cd$ ,  $y_2 = c$ , and  $y_3 = dc$ , and  $K(x_0, \dots, x_n) = K(y_0, \dots, y_n)$ .

Our new alphabet is  $\{c, d\}$ , and we can choose arbitrary numerical values for  $c$  and  $d$ . We want  $y_0$  to be zero-sum, so let  $c = 2$  and  $d = -3$ . Then  $y_1, y_2$  and  $y_3$  are not zero-sum. Lemma 8 shows that  $|K(y_0, \dots, y_n)| \leq 2$ . The idea in Lemma 8 is to compare the number of

occurrences of a certain letter in the words  $\text{psw}(y_0^k)$  and  $\text{psw}(y_1^k \cdots y_n^k)$ . In this particular example, it is sufficient to look at the largest letters: We see that  $\max(\text{psw}(y_0^k)) = 2$  for all  $k$ , but  $\max(\text{psw}(y_1^k y_2^k y_3^k)) \geq \Sigma(y_1^k y_2^k) = k$ . Thus  $y_0^k = y_1^k y_2^k y_3^k$  can hold only for  $k \in \{1, 2\}$ , and it does hold for these two numbers, so  $K(x_0, \dots, x_n) = \{1, 2\}$ .

## 5 Proof of the conjecture

In this section we will prove our main result, Theorem 9. It is preceded by three lemmas: Lemma 6 and Lemma 8 were already mentioned in Section 4, and Lemma 7 is needed in the proof of Lemma 8.

► **Lemma 6.** *Let  $x_0, \dots, x_n \in \Gamma^*$  be zero-sum words. If  $x_0$  is not unary (or equivalently, if  $x_0$  contains a nonzero letter), then there are words  $y_0, \dots, y_n$  such that  $|y_0| < |x_0|$ ,  $K(x_0, \dots, x_n) = K(y_0, \dots, y_n)$ , and  $y_0, \dots, y_n$  commute if and only if  $x_0, \dots, x_n$  commute.*

**Proof.** By Lemma 4, we can let  $Z$  be the basis of the free monoid of zero-sum words over  $\Gamma$ ,  $\Delta$  be an infinite alphabet, and  $h : Z^* \rightarrow \Delta^*$  be an isomorphism. Let  $y_i = h(x_i)$  for all  $i$ . Because  $h$  is an isomorphism, the words  $y_0, \dots, y_n$  satisfy exactly the same equations as  $x_0, \dots, x_n$ . In particular,  $y_0^k = y_1^k \cdots y_n^k$  is equivalent to  $x_0^k = x_1^k \cdots x_n^k$ , and  $y_i y_j = y_j y_i$  is equivalent to  $x_i x_j = x_j x_i$ .

It remains to be shown that  $|y_0| < |x_0|$ . There are words  $z_1, \dots, z_m \in Z$  such that  $x_0 = z_1 \cdots z_m$ . Then  $h(z_i) \in \Delta$  for all  $i$ . The words  $z_i$  cannot be empty, and at least one of them contains a nonzero letter, because  $x_0$  is not unary. This means that at least one of them has length at least 2. Thus  $|y_0| = m < |z_1| + \cdots + |z_m| = |x_0|$ . ◀

In Lemma 8, we will study the words  $\text{psw}(x_0^k)$  and  $\text{psw}(x_1^k \cdots x_n^k)$ . If  $s_i = \Sigma(x_1 \cdots x_{i-1})$  for  $i \in \{1, \dots, n\}$ , then

$$\text{psw}(x_1^k \cdots x_n^k) = \prod_{i=1}^n \text{psw}_{k s_i}(x_i^k),$$

so we will also need to study the words  $\text{psw}_{k s_i}(x_i^k)$  that appear in this product. The following technical lemma will be used to analyze these words.

► **Lemma 7.** *Let  $x \in \Gamma^*$ ,  $s \in \mathbb{R}$ , and  $\Sigma(x) \neq 0$  or  $s \neq 0$ . Let  $k_1, k_2, k_3 \in \mathbb{Z}_+$  and  $k_1 < k_2 < k_3$ . Let  $w_k = \text{psw}_{k s}(x^k)$ . Let  $a \geq \max(w_{k_1} w_{k_2} w_{k_3})$ . Then  $|w_{k_1}|_a \geq |w_{k_2}|_a$ .*

**Proof.** Before the actual proof, let us give the intuitive idea using the geometric concepts in Section 3. The heights of the endpoints of the curve of  $\text{psw}_{k s}(x^k)$  are  $ks$  and  $ks + k\Sigma(x)$ . Their positivity or negativity does not depend on  $k$ , and at least one of them is nonzero by the assumptions. If one of them is positive, then it becomes larger as  $k$  grows, and the highest point on the curve becomes higher as  $k$  grows. This means that  $\max(w_{k_1} w_{k_2} w_{k_3}) = \max(w_{k_3}) > \max(w_{k_2})$ , and thus  $|w_{k_2}|_a = 0$ . If both of the heights of the endpoints are negative, then they become smaller as  $k$  grows, and the highest point on the curve becomes lower as  $k$  grows. This means that  $\max(w_{k_1} w_{k_2} w_{k_3}) = \max(w_{k_1}) > \max(w_{k_2})$ , and thus  $|w_{k_2}|_a = 0$ . If one of the heights of the endpoints is zero and the other is negative, then the highest point on the curve remains the same as  $k$  grows, and also the number of times it occurs remains the same, so  $|w_{k_1}|_a = |w_{k_2}|_a$ .

Let us now move on to the formal proof. For every  $k \in \{k_1, k_2, k_3\}$ , we have

$$w_k = \prod_{i=0}^{k-1} \text{psw}_{k s + i \Sigma(x)}(x) \tag{4}$$



and thus

$$\begin{aligned} \max(w_k) &= \max\{ks + i\Sigma(x) \mid 0 \leq i < k\} + \max(\text{psw}(x)) \\ &= \begin{cases} ks + \max(\text{psw}(x)) & \text{if } \Sigma(x) \leq 0, \\ k(s + \Sigma(x)) - \Sigma(x) + \max(\text{psw}(x)) & \text{if } \Sigma(x) \geq 0. \end{cases} \end{aligned} \quad (5)$$

If  $\Sigma(x) \leq 0$  and  $s \neq 0$  or if  $\Sigma(x) \geq 0$  and  $s + \Sigma(x) \neq 0$ , then (5) is strictly decreasing or strictly increasing with respect to  $k$ , and either  $\max(w_{k_1}) > \max(w_{k_2})$  or  $\max(w_{k_3}) > \max(w_{k_2})$ . In this case,  $a > \max(w_{k_2})$  and thus  $|w_{k_2}|_a = 0$ , which proves the claim.

We still need to consider the case  $\Sigma(x) < 0$  and  $s = 0$ , and the case  $\Sigma(x) > 0$  and  $s + \Sigma(x) = 0$  (if  $\Sigma(x) = 0$ , then  $s = s + \Sigma(x) \neq 0$  by the assumptions). If  $\Sigma(x) < 0$  and  $s = 0$ , then  $a$  can appear in the product (4) only in the term corresponding to  $i = 0$ , which is  $\text{psw}(x)$ . This does not depend on  $k$ , so  $|w_{k_1}|_a = |w_{k_2}|_a$ . Similarly, if  $\Sigma(x) > 0$  and  $s + \Sigma(x) = 0$ , then  $a$  can appear in the product (4) only in the term corresponding to  $i = k - 1$ , which is  $\text{psw}_{-\Sigma(x)}(x)$ . This does not depend on  $k$ , so  $|w_{k_1}|_a = |w_{k_2}|_a$ . This completes the proof.  $\blacktriangleleft$

Words  $u$  and  $v$  are called *abelian equivalent* if  $|u|_a = |v|_a$  for every letter  $a$ . (We could replace abelian equivalence by equality in the next lemma; then the result would be weaker, but still strong enough for our purposes.)

**► Lemma 8.** *Let  $x_0$  be a zero-sum word, and let  $x_1, \dots, x_n$  be words not all of which have zero sum. There are at most two positive integers  $k$  such that  $\text{psw}(x_0^k)$  and  $\text{psw}(x_1^k \cdots x_n^k)$  are abelian equivalent.*

**Proof.** Let  $s_1 = 0$  and  $s_i = \Sigma(x_1 \cdots x_{i-1})$  for  $i \in \{2, \dots, n\}$ . Then

$$\text{psw}(x_1^k \cdots x_n^k) = \prod_{i=1}^n \text{psw}_{k s_i}(x_i^k). \quad (6)$$

Let

$$I_0 = \{i \in \{1, \dots, n\} \mid s_i = 0 \text{ and } \Sigma(x_i) = 0\} \quad \text{and} \quad I_1 = \{1, \dots, n\} \setminus I_0.$$

The set  $I_1$  is nonempty by the assumptions. We have

$$\text{psw}(x_0^k) = \text{psw}(x_0)^k \quad \text{and} \quad \text{psw}_{k s_i}(x_i^k) = \text{psw}(x_i)^k \text{ for all } i \in I_0. \quad (7)$$

Let  $\text{psw}(x_0^k)$  and  $\text{psw}(x_1^k \cdots x_n^k)$  be abelian equivalent for  $k \in \{k_1, k_2, k_3\}$  and  $k_1 < k_2 < k_3$ .

Before the actual proof, let us give the intuitive idea by using the geometric concepts in Section 3. To simplify the explanation, we consider only the case  $x_0 = x_1 \cdots x_n$  here. Because  $x_0$  is zero-sum, the curve of  $\text{psw}(x_0^k)$  consists of  $k$  copies of the curve of  $\text{psw}(x_0)$ , translated horizontally but not vertically. If  $i \in I_0$ , then the curve of  $\text{psw}(x_i)$  appears here  $k$  times, also translated horizontally but not vertically, by the definition of the set  $I_0$ . Similarly, the curve of  $\text{psw}(x_i)$  appears  $k$  times in the curve of  $\text{psw}(x_1^k \cdots x_n^k)$ , again translated horizontally but not vertically. Because we are only interested in abelian equivalence, we can cancel these appearances out (in the formal proof, this corresponds to moving the sum related to  $I_0$  to the left-hand side in (8)). Effectively, this means that we can assume that  $I_0$  is empty. Then, let  $a$  be the largest letter appearing in  $\text{psw}(x_0^k)$ . The letter  $a$  does not depend on  $k$ , and the number of occurrences grows as  $k$  grows. The number of occurrences of  $a$  in  $\text{psw}(x_1^k \cdots x_n^k)$  must be the same, but this leads to a contradiction with Lemma 7.

Let us now move on to the formal proof. For  $k \in \{k_1, k_2, k_3\}$  and every letter  $a$ , we have  $|\text{psw}(x_0^k)|_a = |\text{psw}(x_1^k \cdots x_n^k)|_a$ , so it follows from (6) that

$$|\text{psw}(x_0^k)|_a = \sum_{i=1}^n |\text{psw}_{k s_i}(x_i^k)|_a = \sum_{i \in I_0} |\text{psw}_{k s_i}(x_i^k)|_a + \sum_{i \in I_1} |\text{psw}_{k s_i}(x_i^k)|_a.$$

From (7) it then follows that

$$|\text{psw}(x_0)^k|_a = \sum_{i \in I_0} |\text{psw}(x_i)^k|_a + \sum_{i \in I_1} |\text{psw}_{k s_i}(x_i^k)|_a,$$

and, because  $|w^k|_a = k|w|_a$  for all words  $w$ ,

$$k \left( |\text{psw}(x_0)|_a - \sum_{i \in I_0} |\text{psw}(x_i)|_a \right) = \sum_{i \in I_1} |\text{psw}_{k s_i}(x_i^k)|_a. \quad (8)$$

Consider the largest letter  $a$  which appears in  $\text{psw}_{k s_i}(x_i^k)$  for at least one  $i \in I_1$  and one  $k \in \{k_1, k_2, k_3\}$ . Such a letter must exist, because the set  $I_1$  is not empty. The right-hand side of (8) is positive for this  $a$  and at least one  $k$ , so also the left-hand side must be positive and thus

$$|\text{psw}(x_0)|_a - \sum_{i \in I_0} |\text{psw}(x_i)|_a > 0.$$

But then the left-hand side is positive for all  $k$ , and strictly increasing with respect to  $k$ . For every  $i \in I_1$ , we can use Lemma 7 with  $x = x_i$  and  $s = s_i$ . The assumption  $\Sigma(x) \neq 0$  or  $s \neq 0$  is satisfied because of the definition of  $I_1$ , and the assumption  $a \geq \max(w_{k_1} w_{k_2} w_{k_3})$  is satisfied by the definition of  $a$ . It follows from Lemma 7 that the right-hand side of (8) cannot be larger for  $k_2$  than for  $k_1$ . This contradicts the left-hand side being strictly increasing, and this contradiction proves the claim.  $\blacktriangleleft$

Now we can state as a formal theorem and proof what was said at the beginning of Section 4.

**► Theorem 9.** *Let  $x_0, \dots, x_n \in \Gamma^*$ . If  $x_0^k = x_1^k \cdots x_n^k$  for three positive integers  $k$ , then the words  $x_0, \dots, x_n$  commute.*

**Proof.** We assume that  $x_i x_j \neq x_j x_i$  for some  $i, j$  and prove that  $|K(x_0, \dots, x_n)| \leq 2$ . We can assume that  $x_0$  is minimal in the sense that there does not exist words  $y_0, \dots, y_n$  such that  $y_i y_j \neq y_j y_i$  for some  $i, j$ ,  $K(y_0, \dots, y_n) = K(x_0, \dots, x_n)$ , and  $|y_0| < |x_0|$ . By Lemma 3, we can assume that  $x_0$  is zero-sum.

If  $x_0 \in 0^*$ , then some  $x_i \notin 0^*$  because of the assumption  $x_i x_j \neq x_j x_i$ , and thus  $K(x_0, \dots, x_n) = \emptyset$ . If  $x_0 \notin 0^*$  and  $x_1, \dots, x_n$  are zero-sum, then Lemma 6 contradicts the minimality assumption. If at least one of  $x_1, \dots, x_n$  is not zero-sum, then  $|K(x_0, \dots, x_n)| \leq 2$  by Lemma 8. This completes the proof.  $\blacktriangleleft$

## 6 Conclusion

In this article, we have proved Conjecture 1. One possible direction for further research would be to study equations of the form (2), or some subfamily of these equations (for example, the equations with  $m = 1$ ). Another direction would be to try to apply the methods used in this paper (in particular, Lemma 3, prefix sum words, and the geometric intuition) to some other entirely different problems on word equations. We hope and believe that, in addition to the immediate impact of solving a major open problem, this article will also lead to further advances in the future.

## References

- 1 K.I. Appel and F.M. Djourup. On the equation  $z_1^n z_2^n \cdots z_k^n = y^n$  in a free semigroup. *Trans. Amer. Math. Soc.*, 134:461–470, 1968.
- 2 Srećko Brlek. Interactions between digital geometry and combinatorics on words. In *Proceedings of the 8th WORDS*, volume 63 of *EPTCS*, pages 1–12, 2011. doi:10.4204/EPTCS.63.1.
- 3 Emmanuel Filiot, Olivier Gauwin, Pierre-Alain Reynier, and Frédéric Servais. Streamability of nested word transductions. In *Proceedings of the 31st FSTTCS*, volume 13 of *LIPIcs*, pages 312–324. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2011.
- 4 Ismo Hakala and Juha Kortelainen. On the system of word equations  $x_1^i x_2^i \cdots x_m^i = y_1^i y_2^i \cdots y_n^i$  ( $i = 1, 2, \dots$ ) in a free monoid. *Acta Inform.*, 34(3):217–230, 1997. doi:10.1007/s002360050081.
- 5 Tero Harju and Dirk Nowotka. On the equation  $x^k = z_1^{k_1} z_2^{k_2} \cdots z_n^{k_n}$  in a free semigroup. *Theoret. Comput. Sci.*, 330(1):117–121, 2005. doi:10.1016/j.tcs.2004.09.012.
- 6 Štěpán Holub. A solution of the equation  $(x_1^2 \cdots x_n^2)^3 = (x_1^3 \cdots x_n^3)^2$ . In *Contributions to general algebra, 11 (Olomouc/Velké Karlovice, 1998)*, pages 105–111. Heyn, 1999.
- 7 Štěpán Holub. Local and global cyclicity in free semigroups. *Theoret. Comput. Sci.*, 262(1–2):25–36, 2001. doi:10.1016/S0304-3975(00)00156-0.
- 8 Štěpán Holub and Juha Kortelainen. On systems of word equations with simple loop sets. *Theoret. Comput. Sci.*, 380(3):363–372, 2007. doi:10.1016/j.tcs.2007.03.026.
- 9 Juhani Karhumäki and Wojciech Plandowski. On the defect effect of many identities in free semigroups. In Gheorghe Paun, editor, *Mathematical aspects of natural and formal languages*, pages 225–232. World Scientific, 1994.
- 10 Juha Kortelainen. On the system of word equations  $x_0 u_1^i x_1 u_2^i x_2 \cdots u_m^i x_m = y_0 v_1^i y_1 v_2^i y_2 \cdots v_n^i y_n$  ( $i = 0, 1, 2, \dots$ ) in a free monoid. *J. Autom. Lang. Comb.*, 3(1):43–57, 1998.
- 11 André Lentin. Sur l'équation  $a^M = b^N c^P d^Q$  dans un monoïde libre. *C. R. Acad. Sci. Paris*, 260:3242–3244, 1965.
- 12 M. Lothaire. *Combinatorics on Words*. Addison-Wesley, 1983.
- 13 Roger C. Lyndon and Marcel-Paul Schützenberger. The equation  $a^M = b^N c^P$  in a free group. *Michigan Math. J.*, 9(4):289–298, 1962. doi:10.1307/mmj/1028998766.
- 14 Jarkko Peltomäki and Markus Whiteland. A square root map on Sturmian words (extended abstract). In *Proceedings of the 10th WORDS*, volume 9304 of *LNCS*, pages 197–209. Springer, 2015. doi:10.1007/978-3-319-23660-5\_17.
- 15 Wojciech Plandowski. Test sets for large families of languages. In *Proceedings of the 7th DLT*, volume 2710 of *LNCS*, pages 75–94. Springer, 2003. doi:10.1007/3-540-45007-6\_6.
- 16 Aleksi Saarela. Systems of word equations, polynomials and linear algebra: A new approach. *European J. Combin.*, 47:1–14, 2015. doi:10.1016/j.ejc.2015.01.005.
- 17 Huei Jan Shyr and Shyr-Shen Yu. Non-primitive words in the language  $p^+ q^+$ . *Soochow J. Math.*, 20(4):535–546, 1994.



# Improved Distance Queries and Cycle Counting by Frobenius Normal Form

Piotr Sankowski<sup>\*1</sup> and Karol Węgrzycki<sup>†2</sup>

1 Institute of Informatics, University of Warsaw, Warsaw, Poland  
sank@mimuw.edu.pl

2 Institute of Informatics, University of Warsaw, Warsaw, Poland  
k.wegrzycki@mimuw.edu.pl

---

## Abstract

Consider an unweighted, directed graph  $G$  with the diameter  $D$ . In this paper, we introduce the framework for counting cycles and walks of given length in matrix multiplication time  $\tilde{O}(n^\omega)$ . The framework is based on the fast decomposition into Frobenius normal form and the Hankel matrix-vector multiplication. It allows us to solve the following problems efficiently:

- All Nodes Shortest Cycles – for every node return the length of the shortest cycle containing it. We give an  $\tilde{O}(n^\omega)$  algorithm that improves Yuster [30]  $\tilde{O}(n^{(\omega+3)/2})$  algorithm for unweighted digraphs.
- We show how to compute all  $D$  sets of vertices lying on cycles of length  $c \in \{1, \dots, D\}$  in time  $\tilde{O}(n^\omega)$  randomized time. It improves upon [9] where algorithm that computes a single set is presented.
- We present a functional improvement of distance queries [32] for directed, unweighted graphs.
- All Pairs All Walks – we show almost optimal  $\tilde{O}(n^3)$  time algorithm for all walks counting problem. We improve upon the naive  $O(Dn^\omega)$  time algorithm.

**1998 ACM Subject Classification** F.2.2 Nonnumerical Algorithms and Problems

**Keywords and phrases** Frobenius Normal Form, Graph Algorithms, All Nodes Shortest Cycles

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.56

## 1 Introduction

The *All-Pairs Shortest Paths* (APSP) problem asks to find distances between all pairs of vertices in a graph. For a directed graphs with weights in  $\mathbb{R}$ , there is a classical  $O(n^3)$  time algorithm Floyd and Warshall [12, 28]. Currently best upper bound for this problem is due to Williams [29]  $O(\frac{n^3}{2^{\Omega(\log n)^{0.5}}})$  algorithm. It is asymptotically faster than  $O(n^3/\log^c n)$  for any  $c > 0$  (see survey [6] for earlier algorithms). Showing any algorithm that would work in  $O(n^{3-\epsilon})$  time for some  $\epsilon > 0$  is a major open problem [29].

If we consider unweighted, directed graphs there are subcubic algorithms that exploit fast matrix multiplication. For the undirected graph Seidel [22] presented the optimal  $\tilde{O}(n^\omega)$  time algorithm, where  $\omega < 2.373$  is the matrix multiplication exponent [17]. For the directed case Zwick [33] presented an  $O(n^{2.575})$  time algorithm that is based on the fast rectangular

---

<sup>\*</sup> Piotr Sankowski is supported by the Polish National Science Center, grant no. UMO-2014/13/B/ST6/01811.

<sup>†</sup> The work of Karol Węgrzycki is part of a project TOTAL that has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 677651).



© Piotr Sankowski and Karol Węgrzycki;

licensed under Creative Commons License CC-BY

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 56; pp. 56:1–56:14

Leibniz International Proceedings in Informatics



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

matrix multiplication. Moreover, if we are interested in small integer weights from the set  $\{-M, \dots, M\}$  we have  $O(M^{0.68}n^{2.575})$  algorithm [33].

Because APSP in undirected graphs can be solved in  $\tilde{O}(n^\omega)$ , diameter, radius, shortest cycle, etc. can be determined in  $\tilde{O}(n^\omega)$  time as well. It is surprising that for a directed case, where merely  $O(n^{2.575})$  APSP is known there are also  $\tilde{O}(n^\omega)$  algorithms for determining these properties. After a long line of improvements Cygan et al. [9] showed an  $\tilde{O}(Mn^\omega)$  time algorithms for finding minimum weight perfect matching, shortest cycle, diameter and radius. Also, they showed an application of their techniques that improves upon Yuster [30]  $\tilde{O}(Mn^\omega t)$  time algorithm for the following problem: *determine the set of vertices that lie on some cycle of length at most  $t$* . Cygan et al. [9] managed to solve this problem in  $\tilde{O}(Mn^\omega)$  time using Baur-Strassen's theorem.

All of these algorithms are effective only in the case of a dense graphs. For graphs with the small number of edges there are better algorithms (e.g., APSP in  $\tilde{O}(|V||E|)$  time [26]). But these algorithms are  $\Theta(n^3)$  when  $|E| = \Theta(n^2)$ .

## 2 Related Work

### 2.1 Distance Queries

Yuster and Zwick [32] considered the weighted, directed graphs with weights in  $\{-M, \dots, M\}$ . They showed an algorithm that needs  $\tilde{O}(Mn^\omega)$  preprocessing time. After preprocessing each distance  $\delta(u, v)$  in the graph can be computed exactly in  $O(n)$  query time. In the special case  $M = 1$  they showed  $\tilde{O}(n^\omega)$  algorithm that solves *Single Source Shortest Paths* (SSSP).

We will match their bounds (up to the polylogarithmic factors) using Frobenius normal form. Next we will extend their algorithm so it will return more information about a graph in the same query/preprocessing time.

### 2.2 Counting Cycles

For a given graph  $G$  determining whether  $G$  contains a simple cycle of length exactly  $k$  is NP-hard (in particular determining whether a graph contains a Hamiltonian cycle is NP-complete). However, if we fix  $k$  to be a constant this problem can be solved in polynomial time.

Alon et al. [4] introduced a color coding technique. For a fixed  $k$  if a graph  $G(V, E)$  contains a simple cycle of size exactly  $k$  then such cycle can be found in  $\tilde{O}(|V|^\omega)$  time. Unfortunately, their algorithm depends exponentially  $2^{O(k)}$  on the length of the cycle and in consequence is inapplicable for large  $k$ . In the next years, Alon et al. [5] showed (using a different technique) that for  $k \leq 7$  it is possible to count the number of cycles of length exactly  $k$  in a graph in  $\tilde{O}(|V|^\omega)$  time. In [31] it is shown that for any even  $k$ , cycles of length  $k$  can be found in  $O(|V|^2)$  time in undirected graphs (if they contain such a cycle). Alon et al. [5] showed more methods that depend solely on a number of edges in a graph. For example for odd  $k$  they showed  $O(E^{2-\frac{2}{k+1}})$  algorithm for finding a cycles of length  $k$ . However, for dense graphs these results are worse than Alon et al. [4].

It appears that to break the exponential dependence on the length of the cycle we can do one of the following:

- Consider non-simple cycles (the vertices can reoccur) of length exactly  $k$ ,
- Determine cycles of length at most  $k$ .

To detect whether a non-simple cycle of length exactly  $k$  exists one can use the folklore algorithm. It starts by taking the adjacency matrix  $A$  of a graph  $G$ . Subsequently, in  $\tilde{O}(n^\omega)$

time compute  $A^k$  by repeated squaring. If  $\text{Tr}[A^k] > 0$  then a non-simple  $k$  length cycle exists<sup>1</sup>.

Yuster [30] considered the following problem: *for every vertex in a graph find a shortest cycle that contains it*. He called this problem *All-Nodes Shortest Cycle* (ANSC). He showed a randomized algorithm that solves ANSC for undirected graphs with weights  $\{1, \dots, M\}$  in  $\tilde{O}(\sqrt{M}n^{(\omega+3)/2})$  time. He noted that for simple digraphs (directed graphs with no anti-parallel edges) it is reduced to All-Pairs Shortest Paths problem. The fastest known APSP algorithm for unweighted, directed graphs runs in  $O(n^{2.575})$  due to [33]. Here, we will show how to solve ANSC in  $\tilde{O}(n^\omega)$  for general, unweighted, directed graphs. Unfortunately, our techniques will allow us only to find the length of such a cycle (not determining it). But we can return the set of points, that lie on some cycle of a given length. Independently to our work Agarwal and Ramachandran [3] proved that ANSC can be solved in  $\tilde{O}(n^\omega)$  for unweighted, undirected graphs using a completely different technique.

Yuster [30] also considered following problem: *given a graph and an integer  $t$ . Let  $S(k)$  denote the set of all vertices lying in a cycle of length  $\leq k$ . Determine  $S(t)$* . He considered directed graphs with weights in  $\{-M, \dots, M\}$  and showed  $\tilde{O}(Mn^\omega t)$  algorithm

Recently, Cygan et al. [9] improved his algorithm. They showed that for a fixed  $t \in [0, nM]$  the set  $S(t)$  can be computed in  $\tilde{O}(Mn^\omega)$  randomized time. We show, that for an unweighted ( $M = 1$ ) directed graphs we can compute sets  $S(1), S(2), \dots, S(D)$  in  $\tilde{O}(n^\omega)$  time with high probability.

### 3 Preliminaries

Let  $T(n)$  be the minimal number of algebraic operations needed to compute the product of  $n \times n$  matrix by an  $n \times n$  matrix. We say that  $\omega$  is the exponent of square matrix multiplication. For now the best known upper bound on  $\omega$  is due to Le Gall [17]:

► **Theorem 1** (Le Gall [17]). *For every  $\epsilon > 0$ ,  $T(n) < O(n^{\omega+\epsilon})$  where  $\omega < 2.37287$ .*

In this paper, we will omit  $\epsilon$  in definition and will assume (like in many other papers) that  $O(n^\omega)$  operations are needed to multiply two matrices. The best lower bound for the exponent of matrix multiplication is  $\omega \geq 2$ . For convenience in this paper we will assume that  $\omega > 2$ . The  $\tilde{O}$  notation hides polylogarithmic factors in the complexity. We will use it to emphasize that all our algorithms need the polylogarithmic number of calls to the fast matrix multiplication algorithm.

► **Theorem 2** (Storjohann [24]). *The Frobenius canonical-form of a matrix can be computed deterministically using  $\tilde{O}(n^\omega)$  field operations.*

The comprehensive description of Frobenius normal form will be presented in Section 4. The properties of Frobenius normal form used in this paper are well known in literature [24, 23, 10]. There are also probabilistic algorithms that compute this form in expected  $\tilde{O}(n^\omega)$  with high probability over small fields [11]. In this paper, all algorithms are deterministic if we set the upper bound on the number of distinct walks  $W$ . Then, due to the time of a single field operation we need additional  $O(\log W)$  factor in the complexity. However, if we are only interested if cycle/walk of a given length exists in a graph, we can set sufficiently small field  $\mathbb{Z}_p$  ( $p$  has  $O(\log n)$  bits). This way when algorithm returns nonzero we are sure

<sup>1</sup>  $\text{Tr}[A]$  denotes the trace of a matrix  $A$ , i.e., the sum of elements on main diagonal.

that some walk exists there. If algorithm returns zero, then with high probability there is no such walk.

In this paper, randomization occurs only because for some graphs number of walks can be exponential (e.g.,  $W = O(2^n)$ ) and the output requires to return them.

For matrices  $A \in \mathbb{R}^{n \times k}$  and  $B \in \mathbb{R}^{n \times l}$  the  $A \oplus B \in \mathbb{R}^{n \times (k+l)}$  denotes the concatenation of their columns.  $C_{a,b} \in \mathbb{R}^{n \times (b-a)}$  denotes a matrix constructed by concatenating columns  $c_a, c_{a+1}, \dots, c_b$  of the matrix  $C \in \mathbb{R}^{n \times m}$ .

## 4 Consequences of Frobenius Normal Form

Let  $K$  be a commutative field. For any matrix  $A \in K^{n \times n}$  there exists an invertible  $U$  over  $K$  such that:

$$U^{-1}AU = F = \begin{bmatrix} C_1 & & & 0 \\ & C_2 & & \\ & & C_3 & \\ & & & \ddots \\ 0 & & & & C_k \end{bmatrix}.$$

and  $F$  is the Frobenius-canonical-form<sup>2</sup> of  $A$ . The diagonal block  $C_i$  is called the companion matrix:

$$C_i = \begin{bmatrix} 0 & \dots & 0 & -c_0 \\ 1 & 0 & & -c_1 \\ & 1 & \ddots & \vdots \\ & & \ddots & 0 \\ & & & 1 & 0 & -c_{r-2} \\ 0 & & & & 1 & -c_{r-1} \end{bmatrix} \in K^{r \times r}.$$

Each companion matrix corresponds to the monic polynomial  $C_i(x) = x^r + c_{r-1}x^{r-1} + \dots + c_0 \in K[x]$  and is called the *minimal polynomial* of  $A$ . Each minimal polynomial has a property that  $C_i(A) = 0$ . To guarantee that matrix has only one decomposition into Frobenius normal form we require that every polynomial must divide the next one, i.e.,  $C_i(x) | C_{i+1}(x)$ . The final list of polynomials is called the *invariant factors* of matrix  $A$  [24].

### 4.1 Cyclic Subspaces

Frobenius decomposition can be used to get the desired power of a matrix (analogously to the diagonal decomposition):

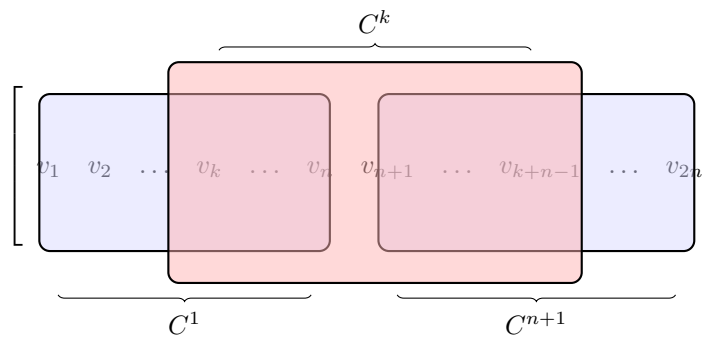
$$A^k = (UFU^{-1})^k = UF(U^{-1}U)F \dots F(U^{-1}U)FU^{-1} = UF^kU^{-1}.$$

Moreover, we will use the property that the power of block diagonal matrix  $F$  is block diagonal:

$$F^k = \begin{bmatrix} C_1^k & & & 0 \\ & C_2^k & & \\ & & C_3^k & \\ & & & \ddots \\ 0 & & & & C_l^k \end{bmatrix}.$$

<sup>2</sup> Sometimes called the rational-canonical form.





■ **Figure 1** Visualisation of the cyclic property (Definition 3).

Now, we need a property of companion matrices that will enable us to power them efficiently.

► **Definition 3** (Cyclic Property). Let  $v_1, \dots, v_n$  be the columns of a matrix  $C \in K^{n \times n}$ . Let  $v_{n+1}, \dots, v_{2n}$  be the columns of matrix  $C^{n+1}$ . If, for every  $1 \leq k \leq n$  the columns of matrix  $C^k$  are  $v_k, v_{k+1}, \dots, v_{k+n}$  then the  $C$  has a **cyclic property**.

► **Theorem 4** (Folklore [14], see [18] for generalization). *Every companion matrix has a cyclic property.*

## 5 Matching Distance Queries on Directed Unweighted Graphs

In this section, we will present a simple algorithm that matches the best known upper bounds of Yuster and Zwick [32] distance queries in directed unweighted graphs.

### 5.1 Answering Distance Queries by Using Frobenius Normal Form

We take the adjacency matrix  $A$  of a graph  $G$  (i.e.,  $n \times n$  matrix with  $a_{u,v} = 1$  when  $(u, v) \in G$  and 0 otherwise). The  $k$ -th power of the adjacency matrix of the graph  $G$  holds the number of walks, i.e., an  $a_{u,v}$  element of  $A^k$  is *the count of distinct walks from  $u$  to  $v$  of length  $k$  in the graph*. The shortest path between vertices  $u, v$  is the smallest  $k$  such that  $A^k$  has nonzero element  $a_{u,v}$ . For a brief moment, we will forget about graph theory interpretation and focus only on finding such  $k$ .

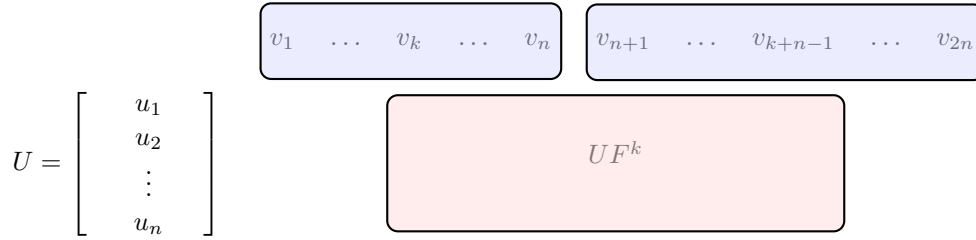
We decompose matrix  $A$  into the Frobenius normal form. Storjohann [24] showed an algorithm that returns  $U$  and  $F$  deterministically in  $\tilde{O}(n^\omega)$  time (note that matrix inverse can also be computed in  $\tilde{O}(n^\omega)$  time).

### 5.2 Single Invariant Factor

To better explain the idea, for a start we will consider a simple case when a number of invariant factors of  $A$  is exactly 1. In that situation, the matrix  $F$  is a companion matrix  $C \in K^{n \times n}$ .

First, we compute the  $(n + 1)$ -th power of the companion matrix  $F^{n+1}$ . This can be done in  $\tilde{O}(n^\omega)$  time by repeated squaring (compute  $F, F^2, F^4, \dots, F^{n+1}$  with  $O(\log n)$  matrix multiplications)<sup>3</sup>.

<sup>3</sup> One can compute  $F^{n+1}$  even faster. Namely, in  $\tilde{O}(n^2)$  time by applying the cyclic property (see Definition 3)



■ **Figure 2** Construction of  $UF^k$  from matrices  $UF$  and  $UF^{n+1}$ .

If the matrix  $UF$  has columns  $v_1, \dots, v_n$  and the matrix  $UF^{n+1}$  has columns  $v_{n+1}, \dots, v_{2n}$ , then the columns  $v_k, \dots, v_{k+n-1}$  construct  $UF^k$  (see Figure 2). It is because the matrix  $F$  has the cyclic property.

This step takes just 2 matrix multiplications, because we need to multiply  $U$  times  $F$  and  $F^{n+1}$ . The preprocessing phase takes only  $\tilde{O}(n^\omega)$  time.

Now, if a user wants to know the number of distinct walks from vertices  $u$  to  $v$  of length exactly  $k$  he takes:

- The  $u$ -th row of matrix  $UF^k$  ( $n$  numbers),
- The  $v$ -th column of matrix  $U^{-1}$ ,
- Multiplies them in  $O(n)$  time (dot product of  $n$  dimensional vectors).

This will give us the  $u, v$  element of matrix  $UF^k U^{-1} = A^k$ . To get the length of the shortest path (i.e., the minimal  $k$  that  $w_{u,v} > 0$ ), we will modify our matrices slightly to get the number of walks of length  $\leq k$ . At the end, we will fit in  $\tilde{O}(n)$  query time (by using binary search) and  $\tilde{O}(n^\omega)$  preprocessing time.

Basically, for a given  $k$  we need to get the  $u, v$  element of matrix  $A + A^2 + \dots + A^k$ . It suffices to add consecutive columns of matrix  $UF \oplus UF^{n+1} = v_1 \oplus v_2 \oplus \dots \oplus v_{2n}$  in the following manner <sup>4</sup>:

$$M' = \begin{bmatrix} v_1 & v_1 + v_2 & v_1 + v_2 + v_3 & \dots & \sum_{i=1}^k v_i & \dots & \sum_{i=1}^{2n} v_i \end{bmatrix} \in \mathbb{R}^{n \times 2n}.$$

Now, to get  $A + A^2 + \dots + A^k$  one needs to multiply  $M'_{k,k+n-1} U^{-1}$  and subtract  $M'_{1,n} U^{-1}$  for a balance <sup>5</sup>.

We can transform matrices  $U$  and  $F$  to matrix  $M'$  in  $O(n^2)$  time during preprocessing. During query, we need to compute 2 dot products ( $u$ -th row of  $M'_{k,k+n-1}$  times  $v$ -th column of  $U^{-1}$  and  $u$ -th row of  $M'_{1,n}$  times  $v$ -th column of  $U^{-1}$ ) and subtract them.

We have an algorithm that for a given vertices  $u, v \in G$  and integer  $k \in \{1, \dots, n\}$  can answer: *how many walks from  $u$  to  $v$  of length less or equal  $k$  are in the graph  $G$*  in  $\tilde{O}(n)$  query and  $\tilde{O}(n^\omega)$  preprocessing.

Because the result of the query is increasing in  $k$  we can use binary search. We can determine the first  $k$  for which the query will answer nonzero value in  $O(\log n)$  tries. Hence, in  $\tilde{O}(n)$  we can find the length of the shortest path. This generalized query can easily return the number of walks of length exactly  $k$ , i.e.,  $q(u, v, k) - q(u, v, k - 1)$ .

For now, we only matched the result of Yuster and Zwick [32] for unweighted graphs with a single, invariant factor. In the next section, we will show how to generalize our technique for

<sup>4</sup> Operation  $\oplus$  denotes concatenation.

<sup>5</sup>  $X_{a,b}$  denotes a matrix constructed by concatenating columns  $x_a, x_{a+1}, \dots, x_b$  of a matrix  $X$ .

graphs with any number of invariant factors. Then, we will extend the Yuster and Zwick [32] algorithm. Namely, we will show that in  $\tilde{O}(n^\omega)$  preprocessing time and  $\tilde{O}(n)$  query time we can get  $D$  numbers (where  $D$  is the graph diameter):  $w_{u,v}^1, w_{u,v}^2, \dots, w_{u,v}^D$ . The number  $w_{u,v}^i$  tells how many walks of length exactly  $i$  are from vertex  $u$  to  $v$ .

With such an algorithm we can easily implement Yuster and Zwick [32] distance queries by linearly scanning (see Section 6). Thus, for the multiple invariant factors we will skip the description of algorithm that returns the number of walks of length smaller than  $k$  (the technique is the same).

### 5.3 Multiple Invariant Factors

Now, we will consider a case when  $k \geq 1$ , i.e., matrix  $F$  has multiple invariant factors. First of all, we need to note that this generalization is not perfect and will allow only the walks of length up to  $D$  (the longest distance in a graph, i.e., diameter).

In the real world applications of our framework (detecting cycles, determining distance between nodes, etc.) we do not need walks longer than the longest possible distance in a graph. It is natural that the diameter is considered to be a bound of an output in graph problems [8, 9, 1, 2].

#### 5.3.1 Relation of the Graph Diameter and Frobenius Normal Form

We begin with relating the graph diameter to the Frobenius normal form. It turns out that the graph diameter is bounded by the degree of a smallest invariant factor.

► **Theorem 5** ([7]). *Given a directed, unweighted graph  $G$  with a diameter  $D$ . Let  $\mu$  denote the degree of the smallest invariant factor (i.e., the dimension of the smallest block in the Frobenius matrix  $F$ ) of an adjacency matrix of the graph  $G$ . Then  $D \leq \mu$ .*

The bounds of this inequality are tight. There are graphs with diameter  $D = \mu$  and graphs with  $\mu = n$  and arbitrary small diameter [7]. Our algorithms are able to return walks up to the length  $\mu$ . We use the bound on  $D$  solely because it is easier to interpret diameter than the *smallest degree of the invariant factor*.

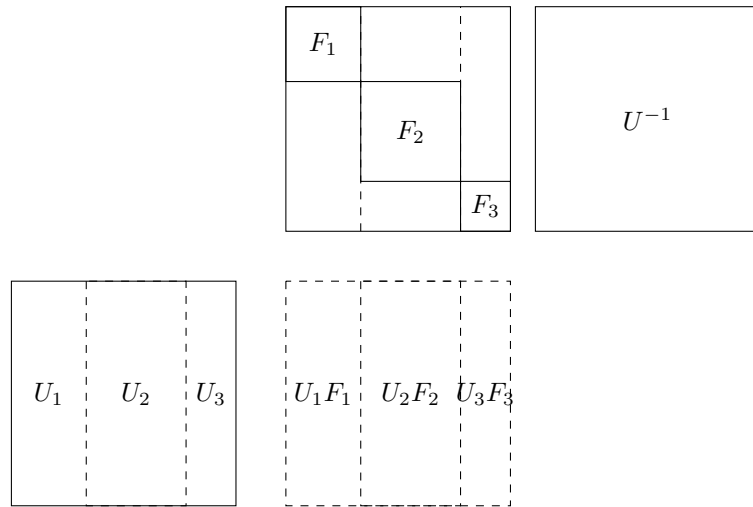
#### 5.3.2 Generalization to Multiple Invariant Factors

Let  $k$  denote the number of blocks in the Frobenius matrix  $F$  and  $\mu$  be the number of columns of the smallest block. To multiply matrix  $U$  by  $F$  we can start by multiplying strips of matrix  $U$  by appropriate blocks of  $F$  and concatenate them later (see Figure 3).

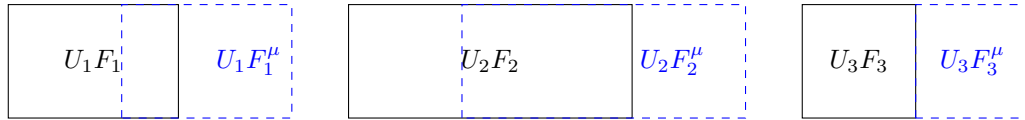
We start by splitting the matrix  $U$  into  $k$  strips with rows corresponding to the appropriate blocks of  $F$  (strip  $U_i$  has as many columns as block  $F_i$ ). Then we multiply  $UF$  and have  $k$  strips:  $U_1F_1, U_2F_2, \dots, U_kF_k$  (each with at least  $\mu$  columns). Next, we multiply  $UF^\mu$  and we keep  $k$  strips:  $U_1F_1^\mu, U_2F_2^\mu, \dots, U_kF_k^\mu$ . Our goal is to get a data structure such that if we need  $UF^k$ , we can quickly choose appropriate columns and append them.

The matrix  $U_iF_i$  has  $l_i$  columns:  $v_1, \dots, v_{l_i}$ . Because  $F_i$  is a companion matrix, the  $U_iF_i^\mu$  has the cyclic property (Definition 3). And the matrix  $U_iF_i^\mu$  has columns:  $v_\mu, \dots, v_{\mu+l_i}$ . There are some duplicate columns in  $U_iF_i$  and  $U_iF_i^\mu$ , when  $\mu < l_i$ . Hence, we only need to keep columns  $v_1, \dots, v_{\mu+l_i}$  for this strip. We do this for all strips (see Figure 4).

We are left with a matrix that has at most  $2n$  columns (because  $l_1 + \mu + l_2 + \mu + \dots + l_k + \mu = k\mu + \sum_{i=1}^n l_i = n + k\mu \leq 2n$ ). To generate it we need to power  $F$  to  $\mu$  and do multiplications  $U \cdot F$  and  $U \cdot F^\mu$ . This can be computed in  $\tilde{O}(n^\omega)$  time.



■ **Figure 3** Multiplication of the block matrix. Example for 3 invariant factors.



■ **Figure 4** Combining strips into a single matrix.

### 5.3.3 Queries with Multiple Invariant Factors

When a query for the number of walks of length  $k$  from node  $u$  to  $v$  comes, we do:

1. For each strip  $i$  take the  $u$ -th row of  $U_i F_i \oplus U_i F_i^\mu$  concatenate them (see Figure 5) into vector  $\bar{u}$ ,
2. Take  $v$ -th column of  $U^{-1}$  matrix and denote it  $\bar{v}$ ,
3. Return the dot product  $\bar{u} \cdot \bar{v}$ .

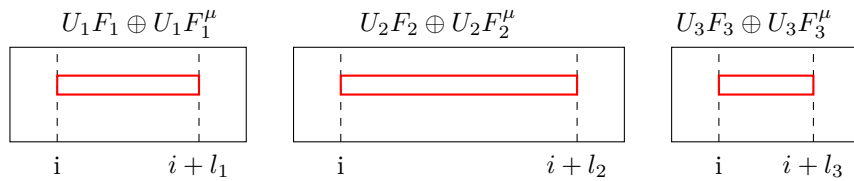
Because  $l_1 + l_2 + \dots + l_k = n$  the vector  $\bar{u} \in K^n$ . Query needs  $O(n)$  time.

Finally, this dot product is  $a_{u,v}$  element of the matrix  $UF^kU^{-1}$ , for a fixed  $k \leq \mu$ . Analogously to Section 5.2 one can extend this result to return the number of walks of length less or equal  $k$ . This matches (up to the polylogarithmic factor) the result of Yuster and Zwick [32]. However in the next section we will show a more general result.

## 6 Almost Optimal Query

In the previous section, we showed how to preprocess a directed, unweighted graph in  $\tilde{O}(n^\omega)$  time in such a way that in  $O(n)$  query we can return a number of distinct walks of a length  $k$  from vertex  $u$  to  $v$ . However, in linear time  $O(n)$  we return only a single number. Our goal is to get far richer information in  $\tilde{O}(n)$  query time.

► **Theorem 6.** *Let  $G = (V, E)$  be a directed, unweighted graph with  $n$  vertices and a diameter  $D$ . There exists an algorithm that after some preprocessing can answer queries for any given  $u, v \in V$ :*



■ **Figure 5** Schema of obtaining vector  $\bar{u}$  for the cycles of length  $i$ .

- Query returns  $\{w_i \mid 1 \leq i \leq D\}$ , where  $w_i$  is the number of distinct walks from  $u$  to  $v$  of length exactly  $i$ ,
- Preprocessing takes  $\tilde{O}(n^\omega)$  and query takes  $\tilde{O}(n)$  field operations.

The algorithm is deterministic (but there are  $O(\log W)$  factors in derandomized version, see Section 3 for explication).

This algorithm is a significant improvement over Yuster and Zwick [32]:

- One can use Theorem 6 to find the distance between  $u, v$  by linearly scanning the array and returning the first  $k$  such that  $w_k > 0$ ,
- Theorem 6 can count cycles. In contrast the Yuster and Zwick [32] cannot, because the distance from  $u$  to  $u$  is always 0 (we will see that in the next section),
- Theorem 6 is almost optimal (when  $D = O(n)$  its output and time are  $O(n)$ ).

On the other hand, Theorem 6 is only a functional improvement and it does not break the  $\tilde{O}(n^\omega)$  of the *Single Source Shortest Paths* (SSSP) for dense, directed graphs.

## 6.1 Hankel Matrix

Now, we will focus on the proof of Theorem 6. First, we need to introduce the Hankel matrix and its properties.

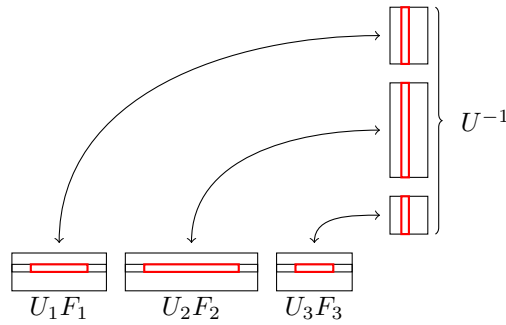
$$H = \begin{pmatrix} c_1 & c_2 & \dots & c_n \\ c_2 & c_3 & \dots & c_{n+1} \\ \vdots & \vdots & & \vdots \\ c_n & c_{n+1} & \dots & c_{2n-1} \end{pmatrix}$$

Hankel matrix is defined by its first row and last column ( $2n - 1$  numbers define  $n \times n$  Hankel matrix). The numbers from the previous row are left-shifted by one and the new number is added at the end. Hankel matrices have some similarities with Topelitz and Circulant matrices.

The basic property we need is that the product of Hankel matrix and vector can be computed in  $O(n \log n)$  time (see [16, 25]) even though explicitly writing the Hankel matrix as  $n \times n$  matrix takes  $O(n^2)$  time. The algorithm takes  $2n - 1$  parameters that define the Hankel matrix and  $n$  parameters that define the vector. The technique is based on the Fast Fourier Transformation [16, 25].

## 6.2 Using Hankel Matrices to Return Number of Walks

The algorithm from Section 5.3.3 concatenates the strips  $U_i F_i$  and builds a single vector. Subsequently, that vector is multiplied by a column of matrix  $U^{-1}$ . But we can also do it in a different order: first we multiply the strip by a section of matrix  $U^{-1}$  and sum the



■ **Figure 6** Multiplication of strips by  $U^{-1}$  matrix. As you can see, matrix  $U^{-1}$  can be split into sections, that multiply only  $U_i F_i$  strips.

results at the end. Thus, we perform  $k$  (number of strips) dot products of smaller vectors (see Figure 6).

Consider the query for a number of walks of length exactly  $k$ . The strips in the matrix  $U^{-1}$  do not depend on  $k$  (vector  $(u_0, \dots, u_l)$ ). However, the vector taken from  $U_i F_i$  (vectors  $(x_i, \dots, x_{i+l})$ ) will be left shifted if we want to compute the next one.

$$\begin{pmatrix} x_0 & x_1 & x_2 & \dots & x_l \\ x_1 & x_2 & \dots & x_l & x_{l+1} \\ x_2 & \dots & x_l & x_{l+1} & x_{l+2} \\ \vdots & & & \vdots & \\ x_\mu & \dots & & x_{\mu+l} & \end{pmatrix} \times \begin{pmatrix} u_0 \\ \vdots \\ u_l \end{pmatrix}$$

As you can see, the subsequent rows can be written as the Hankel matrix (we need to add some zeros and discard certain results to get square matrix, but it will not influence asymptotic complexity). By using the *fast Hankel matrix-vector multiplication* we can compute  $\mu$  values for every strip  $i$  in time  $O(l_i \log l_i)$  ( $l_i$  was defined as the length of  $i$ -th strip). At the end, we need to sum all results into a single array. The total complexity is  $O(\sum_{i=1}^k l_i \log l_i)$ . Because  $\sum_{i=1}^k l_i = n$  the algorithm needs  $O(n \log n)$  field operations. It proves Theorem 6.

We need to address some issues regarding field operations. As mentioned in the Section 3, the  $\tilde{O}$  notation hides the polylogarithmic factors. Here, we silently assume that the number of walks is bounded by  $W$ . It means that the complexity is multiplied by  $O(\log W)$  factor because of the cost of arithmetic operations. If the number of walks is exponential in  $n$  then the cost increases by a linear factor. However, at the beginning we could randomly select the prime number  $p$  with  $O(\log n)$  bits and do arithmetic operations in the field  $\mathbb{Z}_p$ . In some applications we can use our algorithm to answer whether with high probability there exists a walk of a given length. The problem of large number of distinct walks is more of a theoretical issue than a practical one.

### 6.3 All Pairs All Walks (APAW)

Now we will show the application of Theorem 6. We begin with almost optimal algorithm to compute the number of all walks between all pairs of vertices. We are not aware of any other research concerning this problem.

► **Definition 7** (All Pairs All Walks problem). Given a directed, unweighted graph  $G$  with a diameter  $D$ . The task is to return an array  $A$ , such that for every pair of vertices  $u, v \in G$  and every  $k \in \{1, \dots, D\}$  an element  $A[u, v, k]$  is the number of distinct walks from  $u$  to  $v$  of length  $k$ .

The only solution to this problem we are aware of needs  $O(Dn^\omega)$  time. The naive approach: take the adjacency matrix  $A$  of graph  $G$  and save it in  $A[u, v, 1]$ . Then, square it to get  $A^2$  and save it in  $A[u, v, 2]$ . Continue until you fill out complete table. In the worst case this algorithm needs  $D = O(n)$  matrix multiplications, thus its complexity is  $O(Dn^\omega)$ . At the first glance it is surprising that we can improve it to  $\tilde{O}(n^3)$  especially when  $\omega > 2$ .

The  $\tilde{O}(n^3)$  algorithm works as follows. First, preprocess the algorithm from Theorem 6 which takes  $\tilde{O}(n^\omega)$  time. Then, for every pair of vertices  $u, v$  ask a query. A single query takes  $\tilde{O}(n)$  time. Then, save it in the table  $A[u, v]$  (query gives  $D$  numbers  $w_1, \dots, w_D$ , such that  $w_i$  is the number of walks of length  $i$  and save it  $A[u, v, i] := w_i$ ).

Because there are  $O(n^2)$  pairs and for each pair we need  $\tilde{O}(n)$  time, the complexity of our solution is  $\tilde{O}(n^3)$ . The algorithm is almost optimal because the output in the worst case may be  $O(n^3)$ .

## 7 Counting and Determining the Lengths of Cycles

We will use Theorem 6 to solve All-Nodes Shortest Cycles (ANSC) problem efficiently.

► **Theorem 8.** *There exists an algorithm that for a given unweighted digraph  $G$  with a diameter  $D$ :*

- For every vertex  $u$  returns  $D$  numbers:  $c_u^1, c_u^2, \dots, c_u^D$ ,
- The  $c_u^k$  is the number of non-simple cycles of length exactly  $k$ , that contain vertex  $u$ ,
- Algorithm works in  $\tilde{O}(n^\omega \log W)$  time (where  $W$  is the maximum  $c_u^k$ ).

**Proof.** We will use Theorem 6. We start by preprocessing the graph  $G$  in time  $\tilde{O}(n^\omega)$ . Theorem 6 allows us to ask for a number of walks from  $u$  to  $v$  and receive  $D$  numbers:  $w_{u,v}^k$ . So, we ask for the number of walks from vertex  $u$  to the same vertex  $u$ . This is exactly the number of non-simple cycles of a given length that contain vertex  $u$ .

Because we need to ask only  $n$  queries (it is the number of vertices in a graph) and each query takes  $\tilde{O}(n)$  time we have  $\tilde{O}(n^\omega + n^2) = \tilde{O}(n^\omega)$  algorithm. ◀

### 7.1 Solving ANSC Faster

To solve ANSC problem in  $\tilde{O}(n^\omega)$  time and beat Yuster [30]  $\tilde{O}(n^{(\omega+3)/2})$  algorithm we do the linear search on the output. For every vertex we search for the first nonzero element linearly. This with high probability is the length of the shortest cycle that contains it. Because the output contains  $O(n^2)$  numbers the complexity is equal to the preprocessing time  $\tilde{O}(n^\omega)$ .

Similarly, we can scan the output to compute the set  $S(c)$  that contains all vertices that lie on some cycle of length  $\leq c$ . Then, by linear scan we can return the sets  $S(1), \dots, S(D)$ . This improves upon Cygan et al. [9].

## 8 Conclusion and Open Problems

We introduced the framework based on Frobenius normal form and used it to solve problems on directed, unweighted graphs in matrix multiplication time. The main open question is to use this framework to prove that APSP on such graphs can be solved in  $\tilde{O}(n^\omega)$  or at least  $O(n^{2.5})$ . The promising way is to use the algorithms that determine operators of matrices of polynomials (e.g., determinant, solving linear system [19, 15]). Additionally, algorithms for a black-box polynomial degree determination seem to be a promising path.

Another interesting problem is to use this framework to obtain additive approximation for APSP. Currently, the best additive approximation of APSP is due to [21]. However, none of known additive approximation of APSP works in  $\tilde{O}(n^\omega)$  time.

Application in dynamic algorithm also seems to be a promising approach. Frandsen and Sankowski [13] showed an algorithm, that dynamically preserves Frobenius normal form in  $O(kn^2)$  time. Our algorithms use fast Hankel matrix-vector multiplication. The algorithm behind fast Hankel matrix-vector multiplication is based on Discrete Fourier Transform (DFT). Reif and Tate [20] presented an  $O(\sqrt{n})$  time per request algorithm for DFT. Can we use those approaches to obtain a faster dynamic algorithm?

Finally, it remains open how to apply the Frobenius normal form in the weighted directed graphs with small, integer weights  $\{-M, \dots, M\}$ . Cygan et al. [9] took degree  $M$  polynomials and used [19] to get radius and diameter in  $\tilde{O}(Mn^\omega)$  time. We suspect that similar technique can be applied to Frobenius normal form framework.

**Acknowledgment.** We would also like to thank Agnieszka Ciepielewska for meaningful discussions and useful comments.

---

## References

- 1 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1681–1697. SIAM, 2015. doi:10.1137/1.9781611973730.112.
- 2 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. 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 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.
- 3 Udit Agarwal and Vijaya Ramachandran. Fine-grained complexity and conditional hardness for sparse graphs. *arXiv preprint arXiv:1611.07008*, November 2016.
- 4 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.
- 5 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.
- 6 Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 590–598. ACM, 2007. doi:10.1145/1250790.1250877.
- 7 Fan R. K. Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.
- 8 Fan R. K. Chung, V. Faber, and Thomas A. Manteuffel. An upper bound on the diameter of a graph from eigenvalues associated with its laplacian. *SIAM J. Discrete Math.*, 7(3):443–457, 1994. doi:10.1137/S0895480191217776.
- 9 Marek Cygan, Harold N. Gabow, and Piotr Sankowski. Algorithmic applications of baurstrassen’s theorem: Shortest cycles, diameter, and matchings. *J. ACM*, 62(4):28, 2015. doi:10.1145/2736283.
- 10 David Steven Dummit and Richard M. Foote. *Abstract algebra*, volume 3. Wiley Hoboken, 2004.



- 11 Wayne Eberly. Black box frobenius decompositions over small fields. In Traverso [27], pages 106–113. URL: <http://dl.acm.org/citation.cfm?id=345542>, doi:10.1145/345542.345596.
- 12 Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, 1962. doi:10.1145/367766.368168.
- 13 Gudmund Skovbjerg Frandsen and Piotr Sankowski. Dynamic normal forms and dynamic characteristic polynomial. In *International Colloquium on Automata, Languages, and Programming*, pages 434–446. Springer Berlin Heidelberg, 2008.
- 14 F.R. Gantmacher. *The Theory of Matrices, Vol. 1*. Chelsea, 1959.
- 15 Mark Giesbrecht, Michael J. Jacobson Jr., and Arne Storjohann. Algorithms for large integer matrix problems. In Serdar Boztas and Igor E. Shparlinski, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 14th International Symposium, AAECC-14, Melbourne, Australia November 26-30, 2001, Proceedings*, volume 2227 of *Lecture Notes in Computer Science*, pages 297–307. Springer, 2001. doi:10.1007/3-540-45624-4\_31.
- 16 Gene H. Golub and Charles F. Van Loan. *Matrix computations (3. ed.)*. Johns Hopkins University Press, 1996.
- 17 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*, pages 296–303. ACM, 2014.
- 18 Arthur Lim and Jialing Dai. On product of companion matrices. *Linear Algebra and its Applications*, 435(11):2921–2935, 2011.
- 19 Thom Mulders and Arne Storjohann. Rational solutions of singular linear systems. In Traverso [27], pages 242–249. URL: <http://dl.acm.org/citation.cfm?id=345542>, doi:10.1145/345542.345644.
- 20 John H. Reif and Stephen R. Tate. On dynamic algorithms for algebraic problems. *J. Algorithms*, 22(2):347–371, 1997. doi:10.1006/jagm.1995.0807.
- 21 Liam Roditty and Asaf Shapira. All-pairs shortest paths with a sublinear additive error. *ACM Trans. Algorithms*, 7(4):45:1–45:12, September 2011. doi:10.1145/2000807.2000813.
- 22 Raimund Seidel. On the all-pairs-shortest-path problem. In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 745–749. ACM, 1992. doi:10.1145/129712.129784.
- 23 Arne Storjohann. An  $O(n^3)$  algorithm for the frobenius normal form. In Volker Weispfenning and Barry M. Trager, editors, *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation, ISSAC'98, Rostock, Germany, August 13-15, 1998*, pages 101–105. ACM, 1998. URL: <http://dl.acm.org/citation.cfm?id=281508>, doi:10.1145/281508.281570.
- 24 Arne Storjohann. Deterministic computation of the frobenius form. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 368–377. IEEE Computer Society, 2001. URL: <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7601>, doi:10.1109/SFCS.2001.959911.
- 25 Zihui Tang, Ramani Duraiswami, and Nail A. Gumerov. Fast Algorithms to Compute Matrix-Vector Products for Pascal Matrices. *Technical Reports from UMIACS UMIACS-TR-2004-08*, 2004/03/25/ 2004. URL: <http://drum.lib.umd.edu/handle/1903/1338>.
- 26 Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM*, 46(3):362–394, 1999. doi:10.1145/316542.316548.

- 27 Carlo Traverso, editor. *Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation, ISSAC 2000, St. Andrews, United Kingdom, August 6-10, 2000*. ACM, 2000. URL: <http://dl.acm.org/citation.cfm?id=345542>.
- 28 Stephen Warshall. A theorem on boolean matrices. *J. ACM*, 9(1):11–12, 1962. doi:10.1145/321105.321107.
- 29 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 – June 03, 2014*, pages 664–673. ACM, 2014. URL: <http://dl.acm.org/citation.cfm?id=2591796>, doi:10.1145/2591796.2591811.
- 30 Raphael Yuster. A shortest cycle for each vertex of a graph. *Inf. Process. Lett.*, 111(21-22):1057–1061, 2011. doi:10.1016/j.ipl.2011.07.019.
- 31 Raphael Yuster and Uri Zwick. Finding even cycles even faster. In Serge Abiteboul and Eli Shamir, editors, *Automata, Languages and Programming, 21st International Colloquium, ICALP94, Jerusalem, Israel, July 11-14, 1994, Proceedings*, volume 820 of *Lecture Notes in Computer Science*, pages 532–543. Springer, 1994. doi:10.1007/3-540-58201-0\_96.
- 32 Raphael Yuster and Uri Zwick. Answering distance queries in directed graphs using fast matrix multiplication. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 389–396. IEEE Computer Society, 2005. URL: <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=10244>, doi:10.1109/SFCS.2005.20.
- 33 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002. doi:10.1145/567112.567114.

# Lower Bounds on Key Derivation for Square-Friendly Applications\*

Maciej Skorski

IST, Klosterneuburg, Austria  
mskorski@ist.ac.at

---

## Abstract

Security of cryptographic applications is typically defined by security games. The adversary, within certain resources, cannot win with probability much better than 0 (for unpredictability applications, like one-way functions) or much better than  $\frac{1}{2}$  (indistinguishability applications for instance encryption schemes). In so called *squared-friendly applications* the winning probability of the adversary, for different values of the application secret randomness, is not only close to 0 or  $\frac{1}{2}$  on average, but also concentrated in the sense that its second central moment is small. The class of squared-friendly applications, which contains all unpredictability applications and many indistinguishability applications, is particularly important for key derivation. Barak et al. observed that for square-friendly applications one can beat the “RT-bound”, extracting secure keys with significantly smaller entropy loss. In turn Dodis and Yu showed that in squared-friendly applications one can directly use a “weak” key, which has only high entropy, as a secure key.

In this paper we give sharp lower bounds on square security assuming security for “weak” keys. We show that *any* application which is either (a) secure with weak keys or (b) allows for entropy savings for keys derived by universal hashing, *must* be square-friendly. Quantitatively, our lower bounds match the positive results of Dodis and Yu and Barak et al. (TCC’13, CRYPTO’11) Hence, they can be understood as a general characterization of squared-friendly applications.

While the positive results on squared-friendly applications were derived by one clever application of the Cauchy-Schwarz Inequality, for tight lower bounds we need more machinery. In our approach we use convex optimization techniques and some theory of circular matrices.

**1998 ACM Subject Classification** F.2.3 Tradeoffs between Complexity Measures, G.1.6 [Optimization] Convex Programming

**Keywords and phrases** key derivation, square-friendly applications, lower bounds

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.57

## 1 Introduction

When analyzing security of cryptographic primitives one typically assumes access to *perfect* randomness. In practice, we are often limited to *imperfect* sources of randomness. An important research problem is to understand when this “weak” randomness can be used to substitute or extract ideal randomness.

---

\* Supported by the European Research Council consolidator grant (682815-TOCNeT). This is an extended abstract; the full version of the paper appears as ePrint report 2016/157 (<https://eprint.iacr.org/2016/157>).



© Maciej Skorski;

licensed under Creative Commons License CC-BY

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 57; pp. 57:1–57:12

Leibniz International Proceedings in Informatics



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

## 1.1 Key derivation

**Ideal and real settings.** For any cryptographic primitive (like encryption or signatures), which needs a “random”  $m$ -bit string  $R$  to sample the secure key<sup>1</sup>, we compare two different settings:

- (a) ideal setting:  $R$  is *perfectly* random: uniform and independent of any side information available to the attacker
- (b) real settings: there is only an *imperfect* entropy source  $X$  and the secure key  $R$  needs to be derived from  $X$ . The attacker may have some *side information* about  $X$ , in particular the additional randomness used to derive  $R$  from  $X$ .

The security of the primitive is parametrized by  $\epsilon$ , which is the success probability (for so called unpredictability applications) or the advantage (for so called indistinguishability applications) of an attacker with certain resources<sup>2</sup>.

**Generic approach and the entropy loss.** The general way to derive a secure key is to “extract” the randomness from  $X$  by a seeded extractor. In particular, the Leftover Hash Lemma implies that if the min-entropy of  $X$  is at least  $m + L$  then  $H(X), H$ , where  $H$  is randomly chosen function from a universal family, is  $\delta$ -close to uniform with  $\delta = \sqrt{2^{-L}}$ . This means that if an application is  $\epsilon$ -secure for uniform  $R$ , then the same application keyed with  $R = H(X)$ , and even published  $H$ , is  $\epsilon'$ -secure with

$$\epsilon' \leq \epsilon + \sqrt{2^{-L}}. \quad (1)$$

where the entropy loss  $L$  is the difference between the entropy of  $X$  and  $m$ . Note that from Equation 1 it follows that we need  $L = 2\log(1/\epsilon)$  to obtain (roughly) the same security  $\epsilon' = 2\epsilon$ . Unfortunately, if we want the security against *all statistical tests*, this loss is necessary for *any* extractor, as implied by the so called “RT-bound” [6].

**Need for better techniques for cryptographic applications.** The RT-bound does not exclude the possibility of deriving a secure key wasting *much less* than  $2\log(1/\epsilon)$  bits of entropy for *particular* applications. Saving the entropy, apart from scientific curiosity, is a problem of real-world applications. Minimizing the entropy loss is of crucial importance for some applications where it affects efficiency (for example in the elliptic-curve Diffie-Hellman key exchange) and sometimes the entropy amount we have is bounded (e.g. biometric data) than the required length of a key; see also the discussion in [1]. Hence, better techniques than simple extracting are desired. Below we discuss what is known about possible improvements in key derivation for cryptographic applications.

**Key derivation for unpredictability applications.** It is known that unpredictability applications directly tolerate weak keys, provided that the entropy deficiency is not too big. More precisely, any unpredictability application which is  $\epsilon$ -secure with the uniform  $m$ -bit key, is also  $\epsilon' = 2^d\epsilon$ -secure for any key of entropy  $m - d$ . If we have a source  $X$  that has “enough” entropy but its length is too big, we can condense it to a string of length  $m$  with almost full entropy. Essentially, since we achieve a very good condensing rate: any  $X$  of  $m + \log \log(1/\epsilon)$  bits of entropy can be condensed to an  $m$  bit string with the entropy deficiency  $d = 3$  which

<sup>1</sup> In applications like block-ciphers,  $R$  is the key itself. In other applications like RSA encryptions,  $R$  is used to sample public or secret keys. We will simply refer to  $R$  as the key.

<sup>2</sup> For example bounded running time, circuit size or the number of oracle queries.

is  $\epsilon' = 2^3\epsilon$ -close to uniform<sup>3</sup>, we are able to derive a key (roughly) equally secure as the uniform key, with the entropy loss only  $L = O(\log \log(1/\epsilon))$ , i.e. actually without entropy waste [4].

**Key derivation for indistinguishability applications.** The situation for indistinguishability applications is completely different. For the one-time pad which needs an  $m$ -bit uniform key, a key of even  $m - 1$  bits of entropy might be insecure[1]. For some applications we can overcome this difficulty if the winning probability of the adversary, as a function of the key, is not only close to  $1/2$  on average, but also *concentrated* around  $1/2$ . Recall that the advantage of an attacker, for a particular key, is defined as the difference<sup>4</sup> between the winning probability and  $1/2$ . One introduces the following two interesting properties:

- (a) strong security: the *absolute* advantage is small on average (close to the advantage)
- (b) square security: the *squared* advantage is small on average (close to the advantage)

Property (a) provides basically the same bounds as for the unpredictability applications. Namely, we can apply a weak key directly, losing a factor  $2^d$  in the security where  $d$  is the entropy deficiency. Unfortunately, this holds only for a very limited class of applications. Property (b) offers slightly worse bounds but is satisfied for a wide class of indistinguishability applications, called “squared-friendly”. One can use a “weak key” *directly* with a squared-friendly application achieving security of roughly  $\sqrt{2^d}\epsilon$  where  $\epsilon$  is the security with the uniform key and  $d$  is the entropy deficiency [5]. Alternatively, if we want to obtain security  $O(\epsilon)$  instead of  $O(\sqrt{\epsilon})$ , one can use universal hashing to extract an  $\epsilon$ -secure key with the entropy loss reduced by half[1], i.e. up to  $L = \log(1/\epsilon)$ . The improvement in the security analysis over the “standard” Leftover Hash Lemma (LHL) comes from restrictions on the class of the test functions, imposed by the squared-friendly assumption.

## 1.2 Our results

In what follows we assume that  $P$  is an arbitrary indistinguishability application which needs an  $m$ -bit uniform key. We give *tight* lower bounds on the amount of square security (the expected square of the attacker’s advantage) or strong-security (the expected absolute average of the attacker’s advantage) that is necessary for an application to be secure with weak keys, that is keys with entropy deficiency. The notion of entropy here is either the min-entropy or the collision entropy. Collision entropy is less restrictive than min-entropy and is a natural choice to applications involving hash functions, like the LHL<sup>5</sup>. It is equally good for squared-friendly applications as min-entropy. Therefore, as remarked in [5], results for collision entropy are more desired. Nevertheless, we provide bounds for both entropy notions<sup>6</sup>.

**Summary of our contribution.** We characterize squared-friendly applications by their “nice” features. Namely, we show that square-friendly applications are *precisely* those applications which are secure with weak keys or offers improvements in the entropy loss for a key derived

<sup>3</sup> Thus, for condensing we lose incomparably less in the amount than for extracting.

<sup>4</sup> In indistinguishability games, an adversary needs to guess a bit at the end of the game. Since he can flip his answer, any bias indicates that his guess is better than a random answer.

<sup>5</sup> For some applications we may prefer LHL over other extractors because of its simplicity, efficiency and nice algebraic features[1].

<sup>6</sup> Actually collision entropy is more challenging and our observations on strong security are known in folklore, but we study also the min-entropy case for the sake of completeness.

by the LHL. Hence the current state of art is optimal: we cannot do better key derivation than for squared-friendly applications unless we build a theory on stronger than collision entropy requirements for weak keys (which would be in some sense inconvenient because of a natural connection between collision entropy and hash functions).

**Any application secure with weak keys has large square-security.** The following results was proved by Dodis and Yu:

► **Theorem ([5]).** *Applications which are  $\sigma$ -square-secure with the uniform key, i.e. when the averaged squared advantage of any attacker is less than  $\sigma$ , are  $\epsilon = \sqrt{2^d \sigma}$ -secure with any key of collision entropy at least  $m - d$ .*

The following question is therefore natural

**Q:** If  $P$  is secure for all keys of *high (collision or min-) entropy*, how much square-security does it have?

We give an answer in the following two theorems. The first is actually trivial and perhaps known in folklore.

► **Theorem.** *Let  $d \geq 1$ . Suppose that  $P$  is  $\epsilon$ -secure with any key of min-entropy at least  $m - d$ . Then  $P$  is  $\epsilon'$ -strongly secure with  $\epsilon' = O(\epsilon)$ .*

The second one is more interesting

► **Theorem (Informal).** *Let  $d \geq 1$ . If  $P$  is  $\epsilon$ -secure with any key of collision entropy at least  $m - d$ , then  $P$  is  $\sigma$ -square-secure with  $\sigma = O(\epsilon^2)$ .*

The bounds in both cases are tight. Note that if the entropy deficiency  $d$  is bounded then our lower bound perfectly (up to a constant factor) matches the result of Dodis and Yu for any  $P$ .

### Square Security is necessary to improve key derivation by condensing collision entropy.

In the previous paragraph, we discussed the case when the entropy deficiency  $d$  is bounded away from 0. However, sometimes we intentionally extremely condense collision entropy so that this gap is close to 0, to achieve better than  $O(\sqrt{\epsilon})$  security at the price of starting with more than  $m$ -bits of entropy. For  $\epsilon$ -secure square friendly applications one can derive by universal hashing a (roughly)  $\epsilon$ -secure key from any source having  $m + \log(1/\epsilon)$  bits of min-entropy (or even collision entropy) [1]. Let us briefly discuss this result. The proof of the classical Leftover Hash lemma consists of two separate claims:

(a) *Universal hash functions can extremely condense collision entropy.*

(b) *Distributions of extremely high collision entropy are close to uniform.*

More precisely, in the first step one applies a random function from a universal family to “condense” the collision entropy of  $X$  from  $m + L$  bits, where  $L \gg 0$ , to an  $m$ -bit string with  $m - \log(1 + 2^{-L}) \approx m - 2^{-L}$  bits of collision entropy<sup>7</sup>. In the next step one shows that any  $m$ -bit random variable with collision entropy at least  $m - \epsilon^2$  is  $\epsilon$ -close to uniform. Thus,  $L = 2 \log(1/\epsilon)$  is enough to obtain  $\epsilon$ -security. As observed by Barak et al. [1], for  $\epsilon$ -secure applications which are *in addition*  $\epsilon$ -square-secure, it suffices to start with  $m - \epsilon$  bits of the collision entropy in step (b), which reduces by half, i.e. up to  $L = \log(1/\epsilon)$  (comparing to the RT-bound), the entropy loss needed to achieve  $\epsilon$ -closeness.

<sup>7</sup> Conditioned on the choice of the function, which can be thought as a *seed* for the condenser.

► **Theorem** ([1]). *Suppose that  $P$  with a uniform  $m$ -bit key is  $\epsilon$ -secure and  $\sigma$ -square secure. Let  $R$  be any key of collision entropy at least  $m - d$  (possibly given some side information). Then  $P$  keyed with  $R$  is  $\epsilon'$ -secure with  $\epsilon' = \epsilon + \sqrt{\sigma(2^d - 1)}$ , even if the used hash function is published. In particular, for  $\sigma = O(\epsilon)$  and  $d = O(\epsilon)$  we obtain  $\epsilon' = O(\epsilon)$ .*

Applying this to  $R$  being  $X$  condensed by universal hashing we get

► **Corollary** ([1]). *Suppose that  $P$  with a uniform  $m$ -bit key is  $\epsilon$ -secure and  $\sigma$ -square secure (that is, average squared advantage of attackers is not bigger than  $\sigma$ ). Suppose that  $X$  has min-entropy (or collision entropy) at least  $m + L$  and let  $R$  be an  $m$ -bit key derived by universal hashing. Then  $P$  keyed with  $R$  is  $\epsilon'$ -secure with  $\epsilon' = \epsilon + \sqrt{2^{-L}\sigma}$ , even if the used hash function is published. In particular,  $\epsilon' = O(\epsilon)$  for  $\sigma = O(\epsilon)$  and  $L = \log(1/\epsilon)$ .*

The first result motivates the following question about weak keys with the entropy deficiency close to 0.

**Q:** Suppose that an application  $P$  is secure for all keys of *extremely condensed collision entropy*, possibly given side information. How much square-security does  $P$  have?

We give an answer in the following theorem

► **Theorem** (Informal). *Let  $d \ll 1$  and suppose that  $P$  is  $\epsilon$ -secure with all keys of collision entropy at least  $m - d$  (possibly given side information, like the condenser's seed). Then  $P$  is  $\sigma$ -square secure with  $\sigma = O(\max(d, \epsilon^2/d))$ .*

Our theorem, applied for  $d = \epsilon$ , shows the full converse of the observation of Barak et al. A good illustrative example is the case of the Leftover Hash Lemma. As mentioned, universal hash functions condense  $m + \log(1/\epsilon)$  bits of entropy into an  $m$ -bit string with  $m - \epsilon$  bits of entropy. If we use universal hash functions *only as a condenser* (which is exactly how we use them in the LHL), then we have a “black-box” equivalence between distributions of collision entropy at least  $m - \epsilon$  and hashes of distributions having at least  $m + \log(1/\epsilon)$  bits of entropy<sup>8</sup>. It follows then that we want to reduce the entropy loss by half to  $L = \log(1/\epsilon)$  and achieve  $\epsilon$ -security, then our application must be  $\epsilon$ -square secure. This lower bound matches the positive result of Barak et al. [1] and, since it holds for *any* application, can be viewed as a *general* characterization of squared-friendly applications.

**Square security is necessary for reducing the entropy loss in the LHL.** As remarked in the discussion in the previous paragraph, we can *heuristically* identify the set of randomly “hashed” high entropy distributions with the set of distributions of extremely high collision entropy (conditioned on the choice of a hash function as the uniform “seed”). This “equivalence”, is reasonable for the “black-box” use of hash functions. However, it is natural to ask if we can prove it formally. That is, we ask if square security is necessary for improvements in the entropy loss for key derived not by a general “black-box” collision entropy condenser but precisely by *hashing*.

**Q:** Suppose that an application  $P$  is secure for any key derived by applying a randomly chosen (almost) universal hash function to a min-entropy source, even if the hash function is published. Suppose that the entropy loss vs security trade-off is significantly better than (pessimistic) RT-bound. Is  $P$  square-secure?

<sup>8</sup> Because the only information that a general condenser provides is about the entropy in its output.

We give an affirmative answer and a lower bound that (almost) matches the results of [1] for any application.

► **Theorem** (The improved LHL [1] is tight for any application, informal). *Let  $\epsilon = 2^{-(1-\beta)m}$  where  $\beta$  is some small positive number. Then there exists an  $\epsilon$ -universal family  $\mathcal{H}$  of hash functions from  $n$  to  $m$  bits, efficiently commutable and samplable with the use of  $n^2$  uniformly random bits, with the following property: for any application  $P$ , if for every source  $X$  of min-entropy at least  $k = m + \log(1/\epsilon)$  and  $H$  chosen randomly from  $\mathcal{H}$  we have that  $P$  is secure with the key  $H(X)$  and published  $H$ , then  $P$  must be  $\sigma$ -square-secure with  $\sigma = \epsilon^{\frac{1-3\beta/2}{1-\beta}}$ .*

This theorem for  $\beta$  close to 1 (exponential but meaningful security) shows that  $\epsilon^{1-o(1)}$ -square-security is *necessary* for saving  $\log(1/\epsilon)$  bits of entropy in deriving an  $\epsilon$ -secure key by universal hashing (which is almost tight since  $\epsilon$ -square-security is enough).

**Square-security bounds are generally optimal.** The improved bound for applications which are square-secure is generally optimal, as observed in [4]. We provide an alternative proof of this result, using our techniques.

► **Theorem 1** (Square-friendly bounds are generally optimal [4]). *For any  $n$ ,  $k \leq n$ , and  $\delta \in (0, 1)$  there exists an application which has  $\delta$ -square security but for some key of Renyi entropy at least  $k$  it achieves only security  $\epsilon = \Omega\left(\sqrt{2^{n-k}\delta}\right)$ .*

The proof appears in the full version. The advantage of our proof is that it abstracts a general condition for this bound to be satisfied. In particular, similar bounds can be obtained for all so called “strongly secure” applications, where attacker’s advantage is nearly zero except a tiny subset of “weak” keys where the attacker wins with heavy advantage.

### 1.3 Our techniques

Our main technical contribution is an explicit characterization of a distribution which maximizes the expectation of a given function, subject to collision entropy constraints. We show that the worst-case distribution has the shape similar to the function, up to a transform which involves taking a threshold and scaling, as illustrated in Figure 1. We apply this characterization to settings where we want to find the distribution of keys which maximizes the attacker advantage. We stress that with our characterization one can compute *optimal* security with weak keys. Previous works [5, 1] obtained good bounds with the Cauchy-Schwarz inequality only, however these techniques cannot be extended to obtain optimal or lower bounds, as we do.

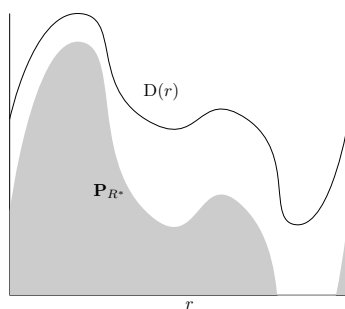
### 1.4 Organization of the paper

In Section 2 we provide the basic notations and definitions for security, square-security and entropy. In Section 3 we discuss the known positive result. Our key auxiliary result on optimization problems with collision entropy constraints is presented in Section 4. The lower bounds are given in Section 5.

## 2 Preliminaries

**Basic notions.** The min entropy of  $X$  is  $\mathbf{H}_\infty(X) = \log(1/\max_x \Pr[X = x])$ . The collision probability of  $X$  is  $\text{CP}(X) = \sum_x \Pr[X = x]^2$ , that is  $\text{CP}(X) = \Pr[X = X']$  where  $X'$  is an





■ **Figure 1** The shape of the best-advantage key distribution (under collision entropy constraints). In application the function  $D(r)$  equals the advantage of an attacker on the key  $r$ .

independent copy of  $X$ . The collision entropy of  $X$  is  $\mathbf{H}_2 = \text{CP}(X)$  and the conditional collision entropy  $\mathbf{H}_2(X|Z)$  equals  $-\log(\mathbf{E}_{z \leftarrow Z} \text{CP}(X|_{Z=z}))$ . The statistical distance of  $X$  and  $Y$  (taking values in the same space) is  $\Delta(X; Y) = \sum_x |\Pr[X = x] - \Pr[Y = x]|$ .

**Security of indistinguishability and unpredictability applications.** Consider any application whose security is defined by a *security game* between an attacker  $A$  and a challenger  $C(r)$ , where  $r$  is an  $m$ -bit key derived from  $U_m$  in the “ideal” setting and from some distribution  $R$  in the “real” setting. For every key  $r$  we denote by  $\text{Win}_A(r)$  the probability (over the randomness used by  $A$  and  $C$ ) that the adversary  $A$  wins the game when challenged on the key  $r$ . The advantage of the adversary  $A$  on the key  $r$  is defined, depending on the type of the application (unpredictability, indistinguishability) as follows:

$$\text{Adv}_A(r) \stackrel{\text{def}}{=} \text{Win}_A(r), \quad (\text{unpredictability}) \quad (2)$$

$$\text{Adv}_A(r) \stackrel{\text{def}}{=} \text{Win}_A(r) - \frac{1}{2}, \quad (\text{indistinguishability}) \quad (3)$$

Now we define the security in the ideal and real models as follows:

► **Definition 2** (Security in the ideal and real model). An application  $\mathcal{P}$  is  $(T, \epsilon)$ -secure in the ideal model if

$$\left| \mathbf{E}_{r \leftarrow U_m} \text{Adv}_A(r) \right| \leq \epsilon \quad (4)$$

for all attackers  $A$  with resources less than  $T$ . We say that  $\mathcal{P}$  is  $(T, \epsilon)$ -secure in the  $(m-d)$ -real<sub>2</sub> if for every distribution  $R$  of collision entropy at least  $m-d$

$$\left| \mathbf{E}_{r \leftarrow R} \text{Adv}_A(r) \right| \leq \epsilon, \quad (5)$$

for all attackers  $A$  with resources less than  $T$ .

► **Remark** (Strong security in the ideal model). If  $\mathbf{E}_{r \leftarrow U_m} |\text{Adv}_A(r)| \leq \epsilon$  in the above setting then we say that  $\mathcal{P}$  is  $(T, \epsilon)$ -strongly secure (in the ideal model).

**Square security.** Finally, we define the notion of square-security (in the ideal model)

► **Definition 3** (Square security). An application  $\mathcal{P}$  is  $(T, \epsilon)$ -square-secure if

$$\mathbf{E}_{r \leftarrow U_m} \text{Adv}_A(r)^2 \leq \epsilon, \quad (6)$$

for all attackers  $A$  with resources less than  $T$ .

**Security in the presence of side information.** Sometimes we need to consider stronger adversaries, which has additional information  $S$ . For example, this is always the case where the weak key has been derived from an entropy source using *public* randomness.

► **Definition 4** (Security in the presence of side information). Given a side information  $S \in \mathcal{S}$ , an application  $\mathcal{P}$  is  $(T, \epsilon)$ -secure in the  $(m - d)$ -real<sub>2</sub> model if for every distribution  $R$  such that  $\mathbf{H}_2(R|S) \geq m - d$  we have

$$\max_{s \in \mathcal{S}} \left| \mathbf{E}_{r \leftarrow R} \text{Adv}_A(r, s) \right| \leq \epsilon, \quad (7)$$

for all attackers  $A$  with resources less than  $T$ . In the ideal model  $\mathcal{P}$  is  $(T, \epsilon)$ -secure if  $\max_{s \in \mathcal{S}} \left| \mathbf{E}_{r \leftarrow U_m} \text{Adv}_A(r, s) \right| \leq \epsilon$  and  $(T, \epsilon)$ -square-secure if  $\max_{s \in \mathcal{S}} \left| \mathbf{E}_{r \leftarrow U_m} \text{Adv}_A(r, s)^2 \right| \leq \epsilon$  for all attackers  $A$  with resources less than  $T$ .

► **Remark.** Note that in the nonuniform setting, security and square security in the ideal model with and without side information coincide.

### 3 Square security – positive results

**Improved key derivation for square-secure applications.** Let  $D$  be an arbitrary real-valued function on  $\{0, 1\}^m$  and let  $Y$  be an arbitrary  $m$ -bit random variable with collision entropy  $\mathbf{H}_2(X) \geq m - d$ . By the Cauchy Schwarz Inequality one obtains [5, 1] the following inequalities

$$\mathbf{E} D(Y) \leq \sqrt{\mathbf{E} D(U_m)^2} \cdot \sqrt{2^d}, \quad (8)$$

$$\mathbf{E} D(Y) - \mathbf{E} D(U_m) \leq \sqrt{\text{Var} D(U_m)} \cdot \sqrt{2^d - 1}. \quad (9)$$

When the side information  $S$  is present, and  $\mathbf{H}_2(Y|S) \geq m - d$ , we get

$$\mathbf{E} D(Y, S) \leq \sqrt{\mathbf{E} D(U_m, S)^2} \cdot \sqrt{2^d}, \quad (10)$$

$$\mathbf{E} D(Y) - \mathbf{E} D(U_Y) \leq \sqrt{\mathbf{E}_{s \leftarrow S} \text{Var} D(U_m, s)} \cdot \sqrt{2^d - 1}. \quad (11)$$

These inequalities, applied to  $D = \text{Adv}_A$  link the security in the real model with the entropy deficiency of a weak key and the security in the ideal model. In particular, one obtains the following results, already mentioned in Section 1.1.

► **Theorem 5** ([5]). *Suppose that  $\mathcal{P}$  is  $(T, \sigma)$ -square secure in the ideal model. Then it is  $(T, \epsilon)$  secure in the  $(m - d)$ -real<sub>2</sub> model with  $\epsilon = \sqrt{2^d} \sigma$ .*

► **Theorem 6** ([1, 5]). *Suppose that an application  $\mathcal{P}$  in the ideal model is  $(T, \epsilon)$ -secure and  $(T, \sigma)$ -square-secure. Then it is  $(T, \delta)$  secure in the real  $(m - d)$ -model with  $\delta = \epsilon + \sqrt{(2^d - 1) \sigma}$ .*

Theorem 5 states that a weak key can be used directly in a square-secure application provided that the entropy deficiency is not too big. The second theorem deals with the case where the deficiency is *extremely* small. It is essentially important when one notices that *universal hash functions* condense collision entropy at a very good rate. Theorem 6 yields the following important corollary

► **Corollary 7** (Improved LHL, [1]). *Suppose that  $\mathcal{P}$  is as above. Let  $X$  be an  $n$ -bit random variable of collision entropy at least  $m + L$ , let  $\mathcal{H}$  be a  $\frac{1+\gamma}{2^m}$ -universal family of functions from  $n$  to  $m$  bits and let  $H$  be a random member of  $\mathcal{H}$ . Then  $\mathcal{P}$  keyed with  $H(X)$  is  $\epsilon'$ -secure with*

$\epsilon' \leq \epsilon + \sqrt{\sigma(\gamma + 2^{-L})}$  against all adversaries with resources  $T$  and given  $H$ . In other words, for all  $A$  with resources  $T$  we have

$$\mathbf{E}_{(r,h) \leftarrow H(X), H} \text{Adv}_A(r, h) \leq \epsilon + \sqrt{\sigma(\gamma + 2^{-L})}. \quad (12)$$

In particular, for  $\gamma \leq \epsilon$  and  $\sigma \leq 4\epsilon$  we achieve security  $\epsilon' \leq 3\epsilon$  with only  $\mathbb{L} = \log(1/\epsilon)$  bits of the entropy loss.

Summing up, when we want to derive a secure key for an  $\epsilon$ -square-secure application from a source  $X$ , we have two options

- (a) We condense (if necessary)  $X$  by hashing into a string with small entropy deficiency. From a source which has  $m - O(1)$  bits of entropy we derive a  $O(\sqrt{\epsilon})$ -secure key.
- (b) If we want more security, we can condense  $X$  even stronger, with deficiency extremely close to 0, sacrificing some entropy amount. From a source which has  $m + \log(1/\epsilon) - O(1)$  bits of entropy we derive a  $O(\epsilon)$ -secure key.

In every case we obtain the meaningful security, in particular even if entropy amount we start with is *smaller* than the length of the key we need. The application of a generic extractor in such a case gives *no* security guarantee! For more examples and applications we refer the reader to [5] and [1].

**Security and square security – mathematical insight.** It is worth of mentioning that the idea behind square security is, conceptually, simple and natural. All we need is the *concentration* of the adversary’s winning probability, which is guaranteed by the small first central or second central moment.

**What applications are square-secure?** It is known that PRGs, PRFs and one-time pads cannot have good square security[2]. In turn, many applications such as stateless chosen plaintext attack (CPA) secure encryption and weak pseudo-random functions (weak PRFs), can be proven to be “square-friendly” that is they have square-security roughly the same as the standard security. The general method to prove that security implies square-security is the so called “double run trick” [1, 5].

## 4 Optimization: auxiliary results

Our main technical tool is a characterization of a distribution that maximizes the expectation of a given function under the collision entropy constraints. It has a nice geometrical interpretation, as the best shape is simply a combination of a threshold and scaling transformation, see Figure 1.

► **Lemma 8** (Maximizing the expectation subject to collision entropy constraints). *Let  $D : S \rightarrow [0, 1]$  be a function on a finite set  $S$  and let  $Y^*$  be any optimal solution to the following problem*

$$\begin{aligned} & \underset{Y}{\text{maximize}} && \mathbf{E} D(Y) \\ & \text{subject to} && \mathbf{H}_2(Y) \geq k \end{aligned} \quad (13)$$

where the maximum is taken over all random variables  $Y$  taking values in  $S$ . Then there exist numbers  $\lambda \geq 0$  and  $t \in \mathbb{R}$  such that  $Y^*$  satisfies the following condition

$$\max(D(x) - t, 0) = \lambda \mathbf{P}_{Y^*}(x) \quad \text{for all } x \in S. \quad (14)$$

## 57:10 Lower Bounds on Key Derivation for Square-Friendly Applications

In particular  $\text{Var}D'(U) = \frac{\lambda^2}{|S|^2}$  where  $D'(x) = \max(D(x) - t, 0)$ . Moreover, if values of  $D(\cdot)$  are all different, then we have  $\lambda > 0$  and  $\lambda, t$  are unique.

► Remark. If values of  $D(\cdot)$  are all different, then  $\lambda > 0$  (see Appendix A).

► **Corollary 9.** We have the following identities  $\mathbf{E} D'(U_m) = \frac{\lambda}{|S|}$ ,  $\mathbf{E} D'(U_m)^2 = \frac{1+\theta}{|S|^2 \lambda^2}$ , and  $\mathbf{E} D'(Y^*) = \frac{(1+\theta)\lambda}{|S|}$  ( $D, \theta, Y^*, \lambda, t$  and  $D'$  are as in Theorem 8).

### 5 Square security – lower bounds

#### 5.1 Weak keys with the entropy deficiency bounded away from 0

We start with the following results, which states that every indistinguishability application which is secure with all keys of high min-entropy must be *strongly secure*. The proof is relatively easy and appears in the full version.

► **Theorem 10.** Suppose that an indistinguishability application  $P$ , which needs an  $m$ -bit key, is  $(T, \epsilon)$ -secure in the  $(m - d)$ - $\text{real}_\infty$  model, for some  $d \geq 1$ . Then  $P$  is  $(T, 2\epsilon)$ -strongly secure. The bound  $2\epsilon$  here is tight.

More challenging and more interesting is the case of an application secure with all keys of high collision entropy.

► **Theorem 11.** Suppose that an indistinguishability application  $P$ , which needs an  $m$ -bit key, is  $(T, \epsilon)$ -secure in the  $(m - d)$ - $\text{real}_2$  model, for some  $d \geq 1$ . Then  $P$  is  $(T, \sigma)$ -square-secure with  $\sigma = 4\epsilon^2$ .

Note that for bounded  $d$  the level of square security perfectly matches the positive result of Dodis and Yu (Theorem 5). We also show (see the end remark in the proof) that this bound is tight up to a constant factor and thus we cannot get the bound  $O(\epsilon^2/2^d)$ , which would exactly match to the positive result in Theorem 5 for all  $d$ . The proof is heavily based on Theorem 8 and appears in the full version.

#### 5.2 Weak keys with the entropy deficiency close to 0

Below we provide a lower bound when the entropy deficiency is close to 0.

► **Theorem 12.** Suppose that  $P$ , which uses an  $m$ -bit key, is  $(T, \epsilon)$ -secure in the  $(m - d)$ - $\text{real}_2$  model (possibly with side information). Then  $P$  is  $\sigma$ -square-secure with  $\sigma \leq \epsilon^2 + \max\left(2^d - 1, \frac{4\epsilon^2}{2^d - 1}\right)$ . In particular, if  $d \ll 1$  then  $\sigma \leq 2 \max(d, \frac{\epsilon^2}{d})$ .

The proof is based on Theorem 8 and appears in the full version. From this we see that Theorem 6 for  $d = \epsilon$  is tight.

#### 5.3 Leftover Hash Lemma as a Key Derivation Function

Finally, we consider the case of a key derived by hashing.

► **Theorem 13.** Let  $\alpha \in [1, 2]$  and let  $\epsilon > 0$ . Suppose that an application  $P$ , which uses an  $m$ -bit secure key, has the following property: for every  $n$ -bit source  $X$  of min-entropy  $k \geq m + \alpha \log(1/\epsilon)$ , and every efficient  $\epsilon^\alpha$ -universal family  $\mathcal{H}$  of hash functions from  $n$  to  $m$  bits, we have

$$\mathbf{E} \text{Adv}_A(H(X), H) \leq C\epsilon,$$

for some constant  $C$  and all adversaries  $A$  with resources at most  $T$ . Then  $P$  is  $(T, \sigma)$ -square-secure with

$$\sigma \leq \frac{3}{2} \cdot \max\left(2^{-m/2}\epsilon^{\alpha/2}, 4(C+1)^2 2^{m/2}\epsilon^{2-\alpha/2}\right). \quad (15)$$

For  $\epsilon > 2^{-m}$  we get  $\sigma = O(2^{m/2}\epsilon^{2-\alpha/2})$ . In particular, if  $\alpha = 1$  and  $\epsilon = 2^{-(1-\beta)m}$  for some  $\beta > 0$  then  $\sigma = O(2^{-(1-3\beta/2)m}) = O\left(\epsilon^{\frac{1-3\beta/2}{1-\beta}}\right)$ . Thus, any application  $P$  which allows deriving an  $\epsilon'$ -secure key with  $\epsilon' = O(\epsilon)$  and entropy loss  $L = \log(1/\epsilon)$  must be  $\sigma = O(\epsilon^{1-o(1)})$ -square-secure. On the positive side we know that  $\sigma$ -square-security with  $\sigma = \epsilon$  is enough (Theorem 7).

► **Corollary 14** (The Improved LHL is tight for any application). *For any application  $P$ , the security guarantee in the improved Leftover Hash Lemma (Theorem 7) cannot be improved by more than a factor  $\epsilon^{o(1)}$ . Note that we require  $\mathcal{H}$  to be efficiently computable and samplable, in order to exclude some (possible) “pathological” counterexamples.*

The proof of Theorem 13 relies on some advanced facts from matrices theory. We briefly sketch our approach, the full proof appears in the full version. The key technical fact we prove is that the hashes of high-min-entropy distributions are really mapped *onto* high collision entropy distributions (with quantitative parameters good enough for our purposes). Once we have a such a correspondence, we reduce the problem to Theorem 12. To this end, we consider the probability  $\Pr[H(x) = y]$  that  $x$  is hashed into  $y$  as a *matrix* with rows  $y$  and columns  $x$  and observe use this matrix to obtain a linear map which realizes that correspondence. To obtain a map with a good behavior, we fill it using some special “pattern” which ensures nice algebraic properties and simplifies inverting.

## 6 Conclusion

We show that the technical condition called *square security* introduced in previous works of Dodis, Yu (TCC’13) and Barak et al. (CRYPTO’11), is not only sufficient but also necessary for better security with weak keys used directly.

---

### References

- 1 B. Barak, Y. Dodis, H. Krawczyk, O. Pereira, K. Pietrzak, F. Standaert, and Yu Yu. Leftover hash lemma, revisited. In *Proc. 31th CRYPTO*, 2011.
- 2 B. Barak, R. Shaltiel, and A. Wigderson. Computational analogues of entropy. In *RANDOM-APPROX*, 2003.
- 3 Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.
- 4 Y. Dodis, K. Pietrzak, and D. Wichs. Key derivation without entropy waste. In *EUROCRYPT*, pages 93–110. Springer Berlin Heidelberg, 2014. doi:10.1007/978-3-642-55220-5\_6.
- 5 Y. Dodis and Yu Yu. Overcoming weak expectations. In *Theory of Cryptography*, volume 7785 of *Lecture Notes in Computer Science*. Springer, 2013. doi:10.1007/978-3-642-36594-2\_1.
- 6 J. Radhakrishnan and A. Ta-Shma. Bounds for dispersers, extractors, and depth-two superconcentrators. *SIAM JOURNAL ON DISCRETE MATHEMATICS*, 13:2000, 2000.

**A Proof of Theorem 8**

**Proof.** Our problem is equivalent to the following constrained maximization problem over  $\mathbb{R}^{|S|}$

$$\begin{aligned}
 & \underset{(p(x))_{x \in \mathbb{R}^{|S|}}}{\text{maximize}} && \sum_x D(x)p(x) \\
 & \text{subject to} && -p(x) \leq 0 \quad \text{for all } x \in S \\
 & && \sum_x p(x) = 1 \\
 & && \sum_x p(x)^2 \leq 2^{-k}
 \end{aligned} \tag{16}$$

The corresponding Lagrangian is given by

$$\begin{aligned}
 L((p(x))_x; (\lambda_1(x))_x, \lambda_2, \lambda_3) = & \sum_x D(x)p(x) + \sum_x \lambda_1(x)p(x) - \lambda_2 \left( \sum_x p(x) - 1 \right) \\
 & - \lambda_3 \left( \sum_x p(x)^2 - 2^{-k} \right)
 \end{aligned} \tag{17}$$

Note that the equality constraint is linear, the inequality constraints are convex and, since  $k < n$ , there exists a vector  $p = p(x)$  such that  $p(x) \geq 0$  for all  $x$ ,  $\sum_x p(x) = 1$  and  $\sum_x p(x)^2 < 2^{-k}$ . This means that Slater's Constraint Qualification is satisfied and the strong duality holds [3]. In this case the Karush-Kuhn-Tucker (KKT) conditions imply that for the optimal solution  $p = p^*$  we have

$$D(x) = -\lambda_1(x) + \lambda_2 + \lambda_3 p^*(x) \tag{18}$$

where  $\lambda_1(x) \geq 0$  for all  $x$ ,  $\lambda_3 \geq 0$  and  $\lambda_2 \in \mathbb{R}$  are the Lagrange Multipliers, satisfying the following so called "complementary slackness" condition

$$\begin{aligned}
 \forall x \quad \lambda_1(x) = 0 & \quad \text{if } p^*(x) > 0, \\
 \lambda_3 = 0 & \quad \text{if } \sum_x (p^*(x))^2 < 2^{-k}.
 \end{aligned} \tag{19}$$

The characterization in Equation 14 follows now by setting  $\lambda = \lambda_3$  and  $t = \lambda_2$ . Indeed, by Equation 18, Equation 19 and  $\lambda_1(x) \geq 0$  we get

$$\max(D(x) - \lambda_2, 0) = \max(-\lambda_1(x) + \lambda_3 p^*(x), 0) = \lambda_3 p^*(x).$$

Finally, note that if all values of  $D(\cdot)$  are different then in Equation 18 we cannot have  $\lambda_3 = 0$ , because then Equation 19 implies that  $D$  is constant on the support of  $p^*$  (which has at least two points provided that  $k > 0$ ). To proof the uniqueness part, observe that if there exists a different pair  $(t', \lambda')$  for the same optimal solution  $p^*$ , then for all  $x$  such that  $p^*(x) > 0$  we have

$$\forall x \in \text{supp}(p^*) \quad D(x) = t + \lambda p^*(x) + t' = \lambda' p^*(x), \tag{20}$$

and, since the case  $\lambda = \lambda'$  cannot happen because it implies  $t = t'$ , we get  $p^*(x) = \frac{t-t'}{\lambda-\lambda'}$ . ◀

# List Approximation for Increasing Kolmogorov Complexity

Marius Zimand

Dept. of Computer and Information Sciences, Towson University, Towson, MD,  
USA

mzimand@towson.edu

---

## Abstract

It is impossible to effectively modify a string in order to increase its Kolmogorov complexity. But is it possible to construct a few strings, not longer than the input string, so that most of them have larger complexity? We show that the answer is yes. We present an algorithm that on input a string  $x$  of length  $n$  returns a list with  $O(n^2)$  many strings, all of length  $n$ , such that 99% of them are more complex than  $x$ , provided the complexity of  $x$  is less than  $n$ . We obtain similar results for other parameters, including a polynomial-time construction.

**1998 ACM Subject Classification** F.1.1 Models of computation, F.2 Analysis of Algorithms and Problem Complexity

**Keywords and phrases** Kolmogorov complexity, list approximation, randomness extractor

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2017.58

## 1 Introduction

The Kolmogorov complexity of a binary string  $x$ , denoted  $C(x)$ , is the minimal description length of  $x$ , i.e., it is the length of a shortest program (in a fixed universal programming system) that prints  $x$ . We analyze the possibility of modifying a string in an effective way in order to obtain a string with higher complexity, without increasing its length. Strings with high complexity exhibit good randomness properties and are potentially useful because they can be employed in lieu of random bits in probabilistic algorithms. It is common to define the randomness deficiency of  $x$  as the difference  $|x| - C(x)$  (where  $|x|$  is the length of  $x$ ), and to say that the smaller the randomness deficiency is, the more random is the string. In this sense, we want to modify a string so that it becomes “more” random. As stated, the above task is impossible because clearly any effective modification cannot increase Kolmogorov complexity (at least not by more than a constant): If  $f$  is a computable function,  $C(f(x)) \leq C(x) + O(1)$ , for every  $x$ . Consequently we have to settle for a weaker solution, and the one we consider is that of list-approximation. List approximation consists in the construction of a list of objects guaranteed to contain at least one element having the desired property. Actually, we try to obtain a stronger type of list approximation, in which, not just one, but *most* of the elements in the list have the desired property. More precisely, we study the following question:

*Question.* Is there a computable function which takes as input a string  $x$  and outputs a short list of strings, which are not longer than  $x$ , such that most of the list’s elements have complexity greater than  $C(x)$ ?

Without the restriction that the length is not increased, the problem is easy to solve by appending a random string (see the discussion in Section 2). The restriction not only makes



© Marius Zimand;

licensed under Creative Commons License CC-BY

34th Symposium on Theoretical Aspects of Computer Science (STACS 2017).

Editors: Heribert Vollmer and Brigitte Vallée; Article No. 58; pp. 58:1–58:12

Leibniz International Proceedings in Informatics



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

the problem interesting, but also amenable to applications in which the input string and the modified strings need to be in a given finite set. The solution that we give can be readily adjusted to handle this case.

The problem of increasing Kolmogorov complexity has been studied before by Buhrman, Fortnow, Newman, and Vereshchagin [3]. They show that there exists a polynomial-time computable  $f$  that on input  $x$  of length  $n$  returns a list of strings, all having length  $n$ , such that if  $C(x) < n$ , then there exists  $y$  in the list with  $C(y) > C(x)$  (this is Theorem 14 in [3]). In the case of complexity conditioned by the string length, they show that it is even possible to compute in polynomial time a list of constant size. That is  $f(x)$  is a list with  $O(1)$ -many strings of length  $n$  and if  $C(x | n) < n$ , then it contains a string  $y$  with  $C(y | n) > C(x | n)$  (this is Theorem 11 in [3]).

As indicated above we are after a stronger type of list approximation: We want on input  $x$  and  $\delta > 0$  to construct a short list of strings not longer than  $x$  with the property that a fraction of  $(1 - \delta)$  of its elements have complexity larger than that of  $x$ . There are several parameters to consider. The first one is the size of the list. The shorter is the list, the better is the approximation. Next, the increasing-complexity procedure that we seek will not work for all strings  $x$ . Recall that  $C(x) \leq |x| + O(1)$  and if  $x$  is a string of maximal complexity at its length, then there simply is no string of larger complexity at its length. In general, for strings  $x$  that have complexity close to  $|x|$ , it is difficult to increase their complexity. Thus, a second parameter is the bound on the complexity of  $x$  for which the increasing-complexity procedure succeeds. The closer this bound is to  $|x|$ , the better is the procedure. The third parameter is the complexity of the procedure. The procedure is required to be computable, but it is preferable if it is computable in polynomial time.

We show the following three results, each one beating the other two with respect to one of these three parameters. The first result exhibits a computable list-approximation for increasing Kolmogorov complexity that works for any  $x$  with complexity  $C(x) < |x|$ .

► **Theorem 1** (Computable list of polynomial size for increasing Kolmogorov complexity). *There exists a computable function  $f$  that on input  $x \in \{0, 1\}^*$  and a rational number  $\delta > 0$ , returns a list of strings of length  $|x|$  with the following properties:*

1. *The size of the list is  $O(|x|^2)\text{poly}(1/\delta)$ ,*
2. *If  $C(x) < |x|$ , then  $(1 - \delta)$  fraction of the elements in the list  $f(x)$  have Kolmogorov complexity larger than  $C(x)$ .*

In the next result, we improve the list size, making it linear in  $|x|$  (for constant  $\delta$ ). The price is that the procedure works only for strings  $x$  with a slightly lower complexity.

► **Theorem 2** (Computable list of linear size for increasing Kolmogorov complexity). *There exists a computable function  $f$  that on input  $x \in \{0, 1\}^*$  and a rational number  $\delta > 0$ , returns a list of strings of length  $|x|$  with the following properties:*

1. *The size of the list is  $O(|x|)\text{poly}(1/\delta)$ ,*
2. *If  $C(x) < |x| - \log |x| - \log \log |x|$ , then  $(1 - \delta)$  fraction of the elements in the list  $f(x)$  have Kolmogorov complexity larger than  $C(x)$ .*

Further reducing the list size remains an interesting open question. We could not establish a lower bound, and, as far as we currently know, it is possible that even constant list size may be achievable.

In the next result, the complexity-increasing procedure runs in polynomial time in the following sense. The size of the list is only quasi-polynomial, but each string in the list is computed in polynomial time.



► **Theorem 3** (Polynomial-time computable list for increasing Kolmogorov complexity). *There exists a function  $f$  that on input  $x \in \{0,1\}^*$  and a constant rational number  $\delta > 0$ , returns a list of strings of length  $|x|$  with the following properties:*

1. *The size of the list is bounded by  $2^{O(\log^3 |x|)}$ ,*
2. *If  $C(x) < |x| - O(\log^3 |x|)$ , then  $(1 - \delta)$  fraction of the elements in the list  $f(x)$  have Kolmogorov complexity larger than  $C(x)$ , and*
3. *The function  $f$  is computable in polynomial time in the following sense: there is a polynomial time algorithm that on input  $x, i$  computes the  $i$ -th element in the list  $f(x)$ .*

Note that the procedure in Theorem 3 can be readily converted into a polynomial-time probabilistic algorithm, which uses  $O(\log^3 |x|)$  random bits to pick at random which element from the list to return.

This paper is inspired by recent list approximation results regarding another problem in Kolmogorov complexity, namely the construction of short programs (or descriptions) for strings. We recall the standard setup for Kolmogorov complexity, which we also use here. We fix an universal Turing machine  $U$ . The universality of  $U$  means that for any Turing machine  $M$ , there exists a computable “translator” function  $t$ , such that for all strings  $p$ ,  $M(p) = U(t(p))$  and  $|t(p)| \leq |p| + O(1)$ . For the polynomial-time constructions we also require that  $t$  is polynomial-time computable. If  $U(p) = x$ , we say that  $p$  is a *program* (or *description*) for  $x$ . The Kolmogorov complexity of the string  $x$  is  $C(x) = \min\{|p| \mid p \text{ is a program for } x\}$ . If  $p$  is a program for  $x$  and  $|p| \leq C(x) + c$ , we say that  $p$  is a  $c$ -short program for  $x$ . Using a Berry paradox argument, it is easy to see that it is impossible to effectively construct a shortest program for  $x$  (or, even a, say,  $n/2$ -short program for  $x$ ). Remarkably, Bauwens et al. [1] show that effective list approximation for short programs is possible: There is an algorithm that, for some constant  $c$ , on input  $x$ , returns a list with  $O(|x|^2)$  many strings guaranteed to contain a  $c$ -short program for  $x$ . They also show a lower bound: The quadratic size of the list is minimal up to constant factors. Teutsch [7] presents a polynomial-time algorithm with similar parameters, except that the list size is larger than quadratic, but still polynomial. The currently shortest list size for a polynomial time list approximation is given by Zimand [10]. Closer to the stronger type of list approximation in this paper, are the probabilistic list approximation results for short programs from Bauwens and Zimand [2] and Zimand [11]. A polynomial-time probabilistic algorithm from [2], on input  $(x, k)$  returns a string  $p$  of length bounded by  $k + O(\log^2 n)$  such that, if the promise  $k = C(x)$  holds, then, with 0.99 probability,  $p$  is a program for  $x$ . In [11], it is shown that the promise can be relaxed to  $k \geq C(x)$ . The survey paper [8] presents most of these results. In this paper, we build on the techniques in [2, 11].

## 2 Techniques and proof overview

We start by explaining why an approach that probably first comes to mind cannot lead to a result with good parameters, such as those obtained in Theorem 1 with a more complicated argument.

Given that we want to modify a string  $x$  so that it becomes more complex, which in a sense means more random, a simple idea is to just append a random string  $z$  to  $x$ . Indeed, if we consider strings  $z$  of length  $c$ , then  $C(xz) > C(x) + c/2$ , for most strings  $z$ , provided  $c$  is large enough. Let us see why this is true. Let  $k = C(x)$  and let  $z$  be a string that satisfies the opposite inequality, that is

$$C(xz) \leq C(x) + c/2. \tag{1}$$

Given a shortest program for  $xz$  and a self-delimited representation of the integer  $c$ , which is  $2 \log c$  bits long, we obtain a description of  $x$  with at most  $k + c/2 + 2 \log c$  bits. Note that from different  $z$ 's satisfying (1), we obtain in this way distinct  $(c/2 + 2 \log c)$ -short programs for  $x$ . By a theorem of Chaitin [4] (also presented as Lemma 3.4.2 in [5]), for any  $d$ , the number of  $d$ -short programs for  $x$  is bounded by  $O(2^d)$ . Thus the number of strings  $z$  satisfying (1) is bounded by  $O(2^{c/2+2 \log c})$ . Since for large  $c$ ,  $O(2^{c/2+2 \log c})$  is much smaller than  $2^c$ , it follows that most strings  $z$  of length  $c$  satisfy the claimed inequality (the opposite of (1)). Therefore, in this way we can obtain a list with a constant number of strings and most of them have complexity larger than  $C(x)$ . The problem with appending a random  $z$  to  $x$ , is that this operation not only increases complexity (which is something we want) but also increases length (which is something we don't want). The natural way to get around this problem is to first compress  $x$  to close to minimal description length using the probabilistic algorithms from [2, 11] described in the Introduction, and then to append  $z$ . However, the algorithms from [2, 11] compress  $x$  to length  $C(x) + O(\log n)$ , where  $n$  is the length of  $x$ . After appending a random  $z$  of length  $c$ , we obtain a string of length  $C(x) + O(\log n) + c$ , and for this to be  $n$  (so that length is not increased), we need  $C(x) \leq n - O(\log n) - c$ . Thus, in this way we cannot obtain a procedure that works for all  $x$  with  $C(x) < n$ , such as the one from Theorem 1.

Our solution is based on a more elaborate construction. The centerpiece is a type of bipartite graph with a low congestion property. Once we have the graph, we view  $x$  as a left node, and the list  $f(x)$  consists of some of the nodes at distance 2 in the graph from  $x$ . (A side remark: Buhrman et al. [3] use graphs as well, namely constant-degree expanders, and they obtain the lists also as the set of neighbors at some given distance.) In our graph, the left side is  $L = \{0, 1\}^n$ , the set of  $n$ -bit strings, the right side is  $R = \{0, 1\}^m$ , the set of  $m$ -bit strings, and each left node has degree  $D$ . The graphs also depend on three parameters  $\epsilon$ ,  $\Delta$ , and  $t$ , and for our discussion it is convenient to also use  $\delta = \epsilon^{1/2}$  and  $s = \delta \cdot \Delta$ . The graphs that we need have two properties. The first one is a low congestion requirement which demands that for every subset  $B$  of left nodes of size at most  $2^t$ ,  $(1 - \delta)$  fraction of nodes in  $B$  share  $(1 - \delta)$  fraction of their right neighbors with at most  $s$  other nodes in  $B$ .<sup>1</sup> The second property is that each right node has at least  $\Delta$  neighbors.

Let us now see how to use such graphs to increase Kolmogorov complexity in the list-approximation sense. Suppose we have a graph  $G$  with the above properties for the parameters  $n, \delta, \Delta, D, s$ , and  $t$ . We claim that for each  $x$  of length  $n$  and with complexity  $C(x) < t$ , we can obtain a list with  $D \cdot \Delta$  many strings, all having length  $n$ , such that at least a fraction of  $(1 - 2\delta)$  of the strings in the list have complexity larger than  $C(x)$ . Indeed, let  $x$  be a string of length  $n$  with  $C(x) = k < t$ . Consider the set  $B = \{x' \in \{0, 1\}^n \mid C(x') \leq k\}$ . Note that the size of  $B$  is bounded by  $2^k$ . A node that does not have the low-congestion property is said to be  $\delta$ -BAD( $B$ ). By the low-congestion of  $G$ , there are at most  $\delta|B|$  elements in  $B$  that are  $\delta$ -BAD( $B$ ). It can be shown that  $x$  is not  $\delta$ -BAD( $B$ ). The reason is, essentially, that the strings that are  $\delta$ -BAD( $B$ ) can be enumerated and they make a small fraction of  $B$  and therefore can be described with less than  $k$  bits. Now, to construct the list, we view  $x$  as a left node in  $G$  and we “go-right-then-go-left.” This means that we first “go-right,” i.e., we take all the  $D$  neighbors of  $x$ , and for each such neighbor  $y$  we “go-left,” i.e., we take  $\Delta$  of the  $y$ 's neighbors and put them in the list. Since  $x$  is not  $\delta$ -BAD( $B$ ),  $(1 - \delta)D$  of its neighbors have at most  $s = \delta \cdot \Delta$  elements in  $B$ . Overall, only  $2\delta \cdot D \cdot \Delta$  of the strings

<sup>1</sup> More formally, for all  $B \subseteq L$  with  $|B| \leq 2^t$ , for all  $x \in B$ , except at most  $\delta|B|$  elements, all neighbors  $y$  of  $x$ , except at most  $\delta D$ , have  $\deg_B(y) \leq s$ , where  $\deg_B(y)$  is the number of  $y$ 's neighbors that are in  $B$ .

in the list can be in  $B$ , and so at least a fraction of  $(1 - 2\delta)$  of the strings in the list have complexity larger than  $k = C(x)$ . Our claim is proved.

For our main results (Theorem 1, Theorem 2, and Theorem 3), we need graphs with the above properties for different settings of parameters. Such graphs can be obtained from randomness extractors, which have been extensively studied in the theory of pseudo-randomness (for example, see Vadhan's monograph [9]). The graphs required by Theorem 1 and Theorem 2 are constructed using the probabilistic method in Lemma 5, and the graph required by Theorem 3 is obtained in Lemma 6 from a randomness extractor of Raz, Reingold, and Vadhan [6].

### 3 Balanced graphs

We define here formally the type of graphs that we need. We work with families of graphs  $G_n = (L, R, E \subseteq L \times R)$ , indexed by  $n$ , which have the following structure:

1. Vertices are labeled with binary strings:  $L = \{0, 1\}^n$ ,  $R = \{0, 1\}^{n-a}$ , where we view  $L$  as the set of left nodes, and  $R$  as the set of right nodes. The parameter  $a$  can be positive or negative, and in absolute value is typically small (less than  $\text{poly}(\log n)$ ).
2. All left nodes have the same degree  $D$ ,  $D = 2^d$  is a power of two, and the edges outgoing from a left node  $x$  are labeled with binary strings of length  $d$ .
3. We allow multiple edges between two nodes. For a node  $x$ , we write  $N(x)$  for the *multiset* of  $x$ 's neighbors, each element being taken with the multiplicity equal to the number of edges from  $x$  landing into it.

A bipartite graph of this type can be viewed as a function  $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^{n-a}$ , where  $\text{EXT}(x, y) = z$  iff there is an edge between  $x$  and  $z$  labeled  $y$ . We want  $\text{EXT}$  to yield a  $(k, \epsilon)$  randomness extractor whenever we consider the modified function  $\text{EXT}_k$  which on input  $(x, y)$  returns  $\text{EXT}(x, y)$  from which we cut the last  $n - k$  bits. Note that the output of  $\text{EXT}_k$  has  $k - a$  bits.

From the function  $\text{EXT}_k$ , we go back to the graph representation, and we obtain the "prefix" bipartite graph  $G_{n,k} = (L = \{0, 1\}^n, R_k = \{0, 1\}^{k-a}, E_k \subseteq L \times R_k)$ , where in  $G_{n,k}$  we merge the right nodes of  $G_n$  that have the same prefix of length  $k - a$ . Since we allow multiple edges between nodes, the left degrees in the prefix graph do not change. However, right degrees may change, and as  $m_k$  gets smaller, right degrees typically get larger due to merging.

The requirement that  $G_{n,k}$  is a  $(k, \epsilon)$  randomness extractor means that for every subset  $B \subseteq L$  of size  $|B| \geq 2^k$ , for every  $A \subseteq R_k$ ,

$$\left| \frac{|E_k(B, A)|}{|B| \times D} - \frac{|A|}{|R_k|} \right| \leq \epsilon, \quad (2)$$

where  $E_k(B, A)$  is the set of edges between  $B$  and  $A$  in  $G_{n,k}$ .

We also want to have the guarantee that each right node in  $G_{n,t}$  has degree at least  $\Delta$ , where  $\Delta$  and  $t$  are parameters.

Accordingly, we have the following definition.

► **Definition 4.** A graph  $G_n = (L, R, E \subseteq L \times R)$  as above is  $(\epsilon, \Delta, t)$ -balanced if the following requirements hold:

1. For every  $k \in \{1, \dots, n\}$ , let  $G_{n,k}$  be the graph corresponding to  $\text{EXT}_k$  described above. We require that, for every  $k \in \{1, \dots, n\}$ ,  $G_{n,k}$  is a  $(k, \epsilon)$  extractor, i.e.,  $G_{n,k}$  has the property in Equation (2).
2. In the graph  $G_{n,t}$ , every right node with non-zero degree has degree at least  $\Delta$ .

In our applications, we need balanced graphs in which the neighbors of a given node can be found effectively, or even in time that is polynomial in  $n$ . As usual, we consider families of graphs  $(G_n)_{n \geq 1}$ , and we say that such a family is *computable* if there is an algorithm that on input  $(x, y)$ , where  $x$  is a left node, and  $y$  is the label of an edge outgoing from  $x$ , outputs  $z$ , where  $z$  is the right node where the edge  $y$  lands. If the algorithm runs in time polynomial in  $n$ , we say that the family  $(G_n)_{n \geq 1}$  is *explicit*. For polynomial-time list approximation, we actually need a stronger property which essentially states that going from right to left can also be done in polynomial time (see the “Moreover...” part in Lemma 6).

The following two lemmas provide the balanced graphs that are used in the proofs of the main result as explained in the proof overview in Section 2.

► **Lemma 5.**

- (a) For every sufficiently large positive integer  $n$ , every rational  $\epsilon > 0$ , and every positive integer constant  $\Delta$ , there is a computable  $(\epsilon, \Delta, t)$ -balanced graph  $G_n = (L = \{0, 1\}^n, R = \{0, 1\}^m, E \subseteq L \times R)$ , with left degree  $D = 2^d = O(n^2 \cdot (1/\epsilon)^2)$ ,  $m = n + 2 \log n$ , and  $t = n$ .
- (b) There exists a constant  $c$ , such that for every sufficiently large positive integer  $n$ , every rational  $\epsilon > 0$ , and every positive integer constant  $\Delta$ , there is a computable  $(\epsilon, \Delta, t)$ -balanced graph  $G_n = (L = \{0, 1\}^n, R = \{0, 1\}^m, E \subseteq L \times R)$ , with left degree  $D = 2^d = O(n \cdot (1/\epsilon)^2)$ ,  $m = n + d - 2 \log(1/\epsilon) - c$ , and  $t = n - \log n - \log \log n$ .

The proof of Lemma 5 is by the standard probabilistic method, and is presented in Section 5.1.

► **Lemma 6.** There exists a constant  $c$  such that for every positive integer  $n$ , every rational  $\epsilon > 0$ , and every positive integer  $\Delta \leq 2^n$ , there is an explicit  $(\epsilon, \Delta, t)$ -balanced graph  $G_n = (L = \{0, 1\}^n, R = \{0, 1\}^m, E \subseteq L \times R)$ , with left degree  $D = 2^d$ , for  $d = O(\log^3(n) \log^2(1/\epsilon))$ ,  $m = n - c \cdot d$ , and  $t = n - \max(0, \log \Delta - c \cdot d)$ .

Moreover, there is an algorithm that on input  $(z, y)$  (and  $n$ ), where  $z \in R = \{0, 1\}^m$  and  $y \in \{0, 1\}^d$  computes a list of  $\Delta$  left neighbors of  $z$  reachable from  $z$  by edges labeled  $y$ , or NIL if there are less than  $\Delta$  such neighbors. This algorithm computes the list implicitly, in the sense that given an index  $i$ , it returns the  $i$ -th element in the list in time polynomial in  $n$  and  $\log i$ .

The proof of Lemma 6 is based on a randomness extractor of Raz, Reingold, and Vadhan [6] and is presented in Section 5.2.

Let us now proceed to the proofs of Theorem 1, Theorem 2, and Theorem 3.

## 4 Proofs of Theorem 1, Theorem 2, and Theorem 3

The theorems have essentially identical proofs, except that balanced graphs with different parameters are used. The following lemma shows a generic transformation of a balanced graph into a function that on input  $x$  produces a list so that most of its elements have complexity larger than  $C(x)$ .

► **Lemma 7.** Suppose that for every constant  $\delta > 0$ , there is  $t = t(n)$ ,  $a = a(n)$ , and a computable (respectively, explicit and satisfying the property stated in the “moreover” part of Lemma 6)  $(\delta^2, (1/\delta)\Delta, t)$ -balanced graph  $G_n = (L_n = \{0, 1\}^n, R_n = \{0, 1\}^{n-a}, E_n \subseteq L_n \times R_n)$ , with  $\Delta = 2(1/\delta^2) \cdot D \cdot 2^a$ , where  $D$  is the left degree.

Then there exists a computable (respectively, polynomial-time computable) function  $f$  that on input a string  $x$  and a rational number  $\delta > 0$  returns a list containing strings of length  $|x|$  and

1. The size of the list is  $2(1/\delta)^3 D^2 2^a$ ,
2. If  $C(x) \leq t$ , then  $(1 - 2\delta)$  of the elements in the list have complexity larger than  $C(x)$ .

**Proof of Lemma 7.** We can assume without loss of generality that  $1/\delta$  is sufficiently large (for the following arguments to be valid) and also that it is a power of 2. Let  $\epsilon = \delta^2$ . Thus,  $\Delta = 2(1/\epsilon) \cdot D \cdot 2^a$ . Let  $x$  be a binary string of length  $n$ , with complexity  $C(x) = k$ . We assume that  $k \leq t$ . We explain how to compute the list  $f(x)$ , with the property stipulated in the theorem's statement.

We take  $G_n$  to be the  $(\epsilon, (1/\delta) \cdot \Delta, t)$ -balanced graph with left nodes of length  $n$  promised by the hypothesis. Let  $G_{n,t}$  be the “prefix” graph obtained from  $G_n$  by cutting the last  $n - t$  bits in the labels of right nodes (thus preserving the prefix of length  $t - a$  in the labels).

The list  $f(x)$  is computed in two steps:

1. First, we view  $x$  as a left node in  $G_{n,t}$  and take  $N(x)$ , the multiset of all neighbors of  $x$  in  $G_{n,t}$ .
2. Secondly, for each  $p$  in  $N(x)$ , we take  $A_p$  to be a set of  $(1/\delta)\Delta$  neighbors of  $p$  in  $G_{n,t}$  (say, the first  $(1/\delta)\Delta$  ones in some canonical order). We set  $f(x) = \bigcup_{p \in N(x)} A_p$  (if  $p$  appears  $n_p$  times in  $N(x)$ , we take  $A_p$  in the union also  $n_p$  times; note that  $f(x)$  is a multiset).

Note that all the elements in the list have length  $n$ , and the size of the list is  $|f(x)| = (1/\delta)\Delta \cdot D = (1/\delta)^3 D^2 2^a$ .

The rest of the proof is dedicated to showing that the list  $f(x)$  satisfies the second item in the statement. Let

$$B_{n,k} = \{x' \in \{0,1\}^n \mid C(x') \leq k\},$$

and let  $S_{n,k} = \lceil \log |B_{n,k}| \rceil$ . Thus,  $2^{S_{n,k}} \leq |B_{n,k}| < 2^{S_{n,k}+1}$ . Later we will use the fact that

$$S_{n,k} \leq k \leq t. \tag{3}$$

We want to use the properties of extractors for sources with min-entropy  $S_{n,k}$  and therefore we consider the graph  $G_{n,S_{n,k}}$ , which is obtained, as we have explained above, from  $G_n$  by taking the prefixes of right nodes of length  $S_{n,k} - a$ . To simplify notation, we use  $G$  instead of  $G_{n,S_{n,k}}$ . The set of left nodes in  $G$  is  $L = \{0,1\}^n$  and the set of right nodes in  $G$  is  $R = \{0,1\}^m$ , for  $m = S_{n,k} - a$ .

We view  $B_{n,k}$  as a subset of the left nodes in  $G$ . Let us introduce some helpful terminology. In the following all the graph concepts (left node, right node, edge, neighbor) refer to the graph  $G$ . We say that a right node  $z$  in  $G$  is  $(1/\epsilon)$ -light if it has at most  $(1/\epsilon) \cdot \frac{|B_{n,k}| \cdot D}{|R|}$  neighbors in  $B_{n,k}$ . A node that is not  $(1/\epsilon)$ -light is said to be  $(1/\epsilon)$ -heavy. Note that

$$(1/\epsilon) \cdot \frac{|B_{n,k}| \cdot D}{|R|} \leq (1/\epsilon) \frac{2^{S_{n,k}+1} \cdot D}{2^{S_{n,k}} \cdot 2^{-a}} = \Delta,$$

and thus an  $(1/\epsilon)$ -light node has at most  $\Delta$  many neighbors in  $B_{n,k}$ .

We also say that a left node in  $B_{n,k}$  is  $\delta$ -BAD with respect to  $B_{n,k}$  if at least a  $\delta$  fraction of the  $D$  edges outgoing from it land in right neighbors that are  $(1/\epsilon)$ -heavy. Let  $\delta$ -BAD( $B_{n,k}$ ) be the set of nodes that are  $\delta$ -BAD with respect to  $B_{n,k}$ .

We show the following claim.

- **Claim 8.** *At most a  $2\delta$  fraction of the nodes in  $B_{n,k}$  are  $\delta$ -BAD with respect to  $B_{n,k}$ . (In other words: for every  $x'$  in  $B_{n,k}$ , except at most a  $2\delta$  fraction, at least a  $(1 - \delta)$  fraction of the edges going out from  $x'$  in  $G$  land in right nodes that have at most  $\Delta$  neighbors with complexity at most  $k$ .)*

We defer for later the proof of Claim 8, and continue the proof of the theorem. For any positive integer  $k$ , let

$$B_k = \{x' \mid C(x') \leq k \text{ and } k \leq t(|x'|)\}.$$

Let  $I_k = \{n \mid k \leq t(n)\}$ . Note that  $|B_k| = \sum_{n \in I_k} |B_{n,k}|$ . Let  $x' \in B_k$ , and let  $n' = |x'|$ . We say that  $x'$  is  $\delta$ -BAD with respect to  $B_k$  if in  $G_{n'}$ ,  $x'$  is  $\delta$ -BAD with respect to  $B_{n',k}$ . We denote  $\delta\text{-BAD}(B_k)$  the set of nodes that are  $\delta$ -BAD with respect to  $B_k$ . We upper bound the size of  $\delta\text{-BAD}(B_k)$ :

$$\begin{aligned} |\delta\text{-BAD}(B_k)| &= \sum_{n' \in I_k} |\delta\text{-BAD}(B_{n',k})| \\ &\leq \sum_{n' \in I_k} 2\delta \cdot |B_{n',k}| \quad (\text{by Claim (8)}) \\ &= 2\delta \sum_{n \in I_k} |B_{n,k}| \\ &= 2\delta |B_k| \\ &\leq 2\delta \cdot 2^{k+1}. \end{aligned}$$

Note that the set  $\delta\text{-BAD}(B_k)$  can be enumerated given  $k$  and  $\delta$ . Therefore a node  $x'$  that is  $\delta$ -BAD with respect to  $B_k$  can be described by  $k$ ,  $\delta$  and its ordinal in the enumeration of the set  $\delta\text{-BAD}(B_k)$ . We write the ordinal on exactly  $k + 2 - \log(1/\delta)$  bits and  $\delta$  in a self-delimited way on  $2 \log \log(1/\delta)$  bits (recall that  $1/\delta$  is a power of 2), so that  $k$  can be inferred from the ordinal and  $\delta$ . It follows that if  $x'$  is  $\delta$ -BAD with respect to  $B_k$ , then, provided  $1/\delta$  is sufficiently large,

$$C(x') \leq k + 2 - \log(1/\delta) + 2 \log \log(1/\delta) + O(1) < k. \tag{4}$$

Now, recall our string  $x \in \{0, 1\}^n$  which has complexity  $C(x) = k$ . The inequality (4) implies that  $x$  cannot be  $\delta$ -BAD with respect to  $B_k$ , which means that  $(1 - \delta)$  of the edges going out from  $x$  land in neighbors in  $G$  having at most  $\Delta$  neighbors in  $B_k$ . The same is true if we replace  $G$  by  $G_{n,t}$ , because, by the inequality (3), right nodes in  $G$  are prefixes of right nodes in  $G_{n,t}$ .

Now suppose we pick at random a neighbor  $p$  of  $x$  in  $G_{n,t}$ , and then find a set  $A_p$  of  $(1/\delta) \cdot \Delta$  neighbors of  $p$  in  $G_{n,t}$ . Then with probability  $1 - \delta$ , only a fraction of  $\delta$  of the elements of  $A_p$  can be in  $B_k$ . Recall that we have defined the list  $f(x)$  to be

$$f(x) = \bigcup_{p \text{ neighbor of } x \text{ in } G_{n,t}} A_p.$$

It follows that  $(1 - 2\delta)$  of elements in  $f(x)$  have complexity larger than  $C(x)$  and this ends the proof. ◀

It remains to prove Claim 8.

**Proof of Claim 8.** Let  $A$  be the set of right nodes that are  $(1/\epsilon)$ -heavy. Then

$$|A| \leq \epsilon |R|.$$

Indeed the number of edges between  $B_{n,k}$  and  $A$  is at least  $|A| \cdot (1/\epsilon) \cdot \frac{|B_{n,k}| \cdot D}{|R|}$  (by the definition of  $(1/\epsilon)$ -heavy), but at the same time the total number of edges between  $B_{n,k}$  and  $R$  is  $|B_{n,k}| \cdot D$  (because each left node has degree  $D$ ).

Next we show that

$$|\delta\text{-BAD}(B_{n,k})| \leq 2\delta |B_{n,k}|. \tag{5}$$

For this, note that  $G$  is a  $(S_{n,k}, \epsilon)$  randomness extractor and  $B_{n,k}$  has size at least  $2^{S_{n,k}}$ . Therefore by the property (2) of extractors,

$$\frac{|E(B_{n,k}, A)|}{|B_{n,k}| \cdot D} \leq \frac{|A|}{|R|} + \epsilon \leq 2\epsilon.$$

On the other hand the number of edges linking  $B_{n,k}$  and  $A$  is at least the number of edges linking  $\delta$ -BAD( $B_{n,k}$ ) and  $A$  and this number is at least  $|\delta$ -BAD( $B_{n,k}$ )  $\cdot \delta D$ . Thus,

$$|E(B_{n,k}, A)| \geq |\delta$$
-BAD( $B_{n,k}$ )|  $\cdot \delta D$ .

Combining the last two inequalities, we obtain

$$\frac{|\delta$$
-BAD( $B_{n,k}$ )|}{|B\_{n,k}|} \leq 2\epsilon \cdot \frac{1}{\delta} = 2\delta.

*End of the proofs of Claim 8 and of Lemma 7.* ◀

Theorem 1, Theorem 2, and Theorem 3 are obtained by plugging into the above lemma the balanced graphs from Lemma 5 and Lemma 6. More precisely, Theorem 1 is obtained by using Lemma 7 with the balanced graph promised by Lemma 5(a), with parameters  $\epsilon = \delta^2$ , and  $\Delta = 2(1/\delta^2) \cdot D \cdot 2^{-2 \log n} = O(1)$ . Theorem 2 is obtained by using Lemma 7 with the balanced graph promised by Lemma 5(b), with parameters  $\epsilon = \delta^2$ , and  $\Delta = 2(1/\delta^2) \cdot D \cdot 2^{-(d-2 \log(1/\epsilon)-c)} = O(1)$ . Finally, Theorem 3 is obtained by using Lemma 7 with the balanced graph promised by Lemma 6, with parameters  $\epsilon = \delta^2$ , and  $\Delta = 2(1/\delta^2) \cdot D \cdot 2^{cd} = 2^{O(\log^3 n)}$ .

## 5 Construction of balanced graphs

### 5.1 Proof of Lemma 5

We prove part (b), where the relations between parameters are somewhat tighter. The proof of part (a) is similar. We use the probabilistic method. For some constant  $c$  that will be fixed later, we consider a random function  $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^{n+d-2 \log(1/\epsilon)-c}$ . We show the following two claims, which imply that a random function has the desired properties with positive probability. Since the properties can be checked effectively, we can find a graph as stipulated in part (b) by exhaustive search. We use the notation from Definition 4 and from the paragraph preceding it.

► **Claim 9.** *For some constant  $c$ , with probability  $\geq 3/4$ , it holds that for every  $k \in \{1, \dots, n\}$ , in the bipartite graph  $G_{n,k} = \{L, R_k, E_k \subseteq L \times R_k\}$ , every  $B \subseteq L = \{0, 1\}^n$  of size  $|B| \geq 2^k$ , and every  $A \subseteq R_k = \{0, 1\}^{k+d-2 \log(1/\epsilon)-c}$  satisfy*

$$\left| \frac{|E_k(B, A)|}{|B| \times D} - \frac{|A|}{|R_k|} \right| \leq \epsilon. \tag{6}$$

► **Claim 10.** *For every sufficiently large positive integer  $n$ , with probability  $\geq 3/4$ , every right node in the graph  $G_{n, n-\log n-\log \log n}$  has at least  $\Delta$  neighbors in  $L$ .*

**Proof of Claim 9.** First we fix  $k \in \{1, \dots, n\}$  and let  $K = 2^k$  and  $N = 2^n$ . Let us consider  $B \subseteq \{0, 1\}^n$  of size  $|B| \geq K$ , and  $A \subseteq R_k$ . For a fixed  $x \in B$  and  $y \in \{0, 1\}^d$ , the probability that  $\text{EXT}_k(x, y)$  is in  $A$  is  $|A|/|R_k|$ . By the Chernoff bounds,

$$\text{Prob} \left[ \left| \frac{|E_k(B, A)|}{|B| \times D} - \frac{|A|}{|R_k|} \right| > \epsilon \right] \leq 2^{-\Omega(K \cdot D \cdot \epsilon^2)}.$$

The probability that relation (6) fails for a fixed  $k$ , some  $B \subseteq \{0, 1\}^k$  of size  $|B| \geq K$  and some  $A \subseteq R_k$  is bounded by  $2^{K \cdot D \cdot \epsilon^2 \cdot 2^{-c}} \cdot \binom{N}{K} \cdot 2^{-\Omega(K \cdot D \cdot \epsilon^2)}$ , because  $A$  can be chosen in  $2^{K \cdot D \cdot \epsilon^2 \cdot 2^{-c}}$  ways, and we can consider that  $B$  has size exactly  $K$  and there are  $\binom{N}{K}$  possible choices of such  $B$ 's. If  $D = \Omega((n - k)/\epsilon^2)$  and  $c$  is sufficiently large, the above probability is much less than  $(1/4)2^{-k}$ . Therefore the probability that relation (6) fails for some  $k$ , some  $B$  and some  $A$  is less than  $1/4$ .  $\blacktriangleleft$

**Proof of Claim 10.** We use a standard ‘‘coupon collector’’ argument. Let  $t = n - \log n - \log \log n$ . Let  $N = 2^n$  and  $C = 2^c$ , where  $c$  is the constant for which Claim 9 holds. We work in the bipartite graph  $G_{n, n - \log n - \log \log n} = (L, R, E \subseteq L \times R)$  in which every left node has degree  $D = 2^d$ ,  $L = \{0, 1\}^n$ , and  $R = \{0, 1\}^m$ , where  $m = (n - \log n - \log \log n + d - 2 \log(1/\epsilon) - c)$ . For a left node  $x$ , an edge labeled  $y \in \{0, 1\}^d$  and a right node  $z$ , we say that  $(x, y)$  hits  $z$  if the  $y$ -labeled edge outgoing from  $x$  lands in  $z$ . We want to show that with high probability each  $z$  is hit at least  $\Delta$  times. Let us order  $\{0, 1\}^n \times \{0, 1\}^d$  in, say, lexicographical order  $\{(x, y)_1 < (x, y)_2 < \dots < (x, y)_{ND}\}$ . We define  $\Delta$  groups of ‘‘shooting’’ at  $R$  by taking  $(x, y)_1, \dots, (x, y)_r$  in the first group,  $(x, y)_{r+1}, \dots, (x, y)_{2r}$  in the second group, and so on with  $r$  left nodes in each group, where  $r$  will be fixed later. The probability that a fixed  $z$  is not hit by some  $(x, y)_i$  is  $(1 - 1/|R|) \leq e^{-1/|R|}$ . The probability that a fixed  $z$  is not hit by any element in a given group is at most  $e^{-r/|R|}$  and the probability that there exists some  $z \in R$  that is not hit by a given group is bounded by  $|R|e^{-r/|R|}$ . We take  $r = |R|(\ln |R| + \ln(4\Delta))$ , and the above probability is bounded by  $1/(4\Delta)$ . Therefore, the probability that some  $z$  in  $R$  is not hit by some group in the set of  $\Delta$  groups is at most  $1/4$ . Note that  $r \cdot \Delta \leq ND$ , provided  $n$  is large enough, and thus all the groups fit into  $\{0, 1\}^n \times \{0, 1\}^d$ .  $\blacktriangleleft$

*End of the proof of Lemma 5.*  $\blacktriangleleft$

## 5.2 Proof of Lemma 6.

The construction relies on the randomness extractor of Raz, Reingold, and Vadhan [6].

► **Theorem 11** (Theorem 22, (2) in [6]). *There exists a function  $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ , computable in time polynomial in  $n$ , with the following properties:*

- (1)  $d = O(\log^3(n) \log^2(1/\epsilon))$ ,
- (2)  $m = n - c \cdot d$ , for some constant  $c$ ,
- (3) For every  $k \leq n$ , the function  $\text{EXT}_k$  obtained by computing  $\text{EXT}$  and cutting the last  $n - k$  bits of the output is a  $(k, \epsilon)$  extractor,
- (4) For every  $y \in \{0, 1\}^d$ , the function  $f_y(x) = \text{EXT}(x, y)$  is a linear function from  $(\text{GF}[2])^n$  to  $(\text{GF}[2])^m$  (where we view  $x \in \{0, 1\}^n$  as an element of  $(\text{GF}[2])^n$  in the natural way). In other words,  $\text{EXT}(x, y) = A_y \cdot x$ , where  $A_y$  is an  $m$ -by- $n$  matrix with entries in  $\text{GF}[2]$ , computable from  $y$  in time polynomial in  $n$ .

*Note.* Item (4) is not explicitly stated in [6], so we provide here a short explanation. The construction given in [6] of  $\text{EXT} : \{0, 1\}^n \times \{0, 1\}^d \rightarrow \{0, 1\}^m$ , views  $x$  as the specification of a function  $u_x(\cdot, \cdot)$  of two variables (in a way that we present below), defines some functions  $g_1(y), h_1(y), \dots, g_m(y), h_m(y)$ , each one computable in time polynomial in  $n$ , and then sets

$$\text{EXT}(x, y) = u_x(g_1(y), h_1(y)), \dots, u_x(g_m(y), h_m(y)), \quad (7)$$

i.e., the  $i$ -th bit is  $u_x(g_i(y), h_i(y))$ . Thus, it is enough to check that  $f_{v,w}(x) = u_x(v, w)$  is linear in  $x$ . Let us now describe  $u_x$ . The characteristic sequence of  $u_x$  is the Reed-Solomon code of  $x$ . More precisely, for some  $s$ ,  $x$  is viewed as a polynomial  $p_x$  over the field  $\text{GF}[2^s]$ .



The elements of  $\text{GF}[2^s]$  are viewed as  $s$ -dimensional vectors over  $\text{GF}[2]$  in the natural way. Note that in this view the evaluation of  $p_x$  at point  $v$  is a linear transformation of  $x$ , i.e.,  $p_x(v) = A_v x$  for some  $s$ -by- $n$  matrix  $A_v$  with entries from  $\text{GF}[2]$ . Finally,  $u_x(v, w)$  is defined as the inner product  $w \cdot p_x(v)$  and therefore  $u_x(v, w) = (w A_v) x$ , and thus it is a linear function in  $x$ . Now we plug  $h_i(y)$  as  $w$  and  $g_i(y)$  as  $v$ , and we build the matrix  $A_y$ , by taking its  $i$ -th row to be  $h_i(y) A_{g_i(y)}$ . Using the Equation (7), we obtain item (4) in the theorem.

Now let us proceed to the actual proof of Lemma 6. The function EXT from Theorem 11 defines the explicit bipartite graph  $G_n$ . Let  $t = n - \max(0, \log \Delta - c \cdot d)$ . By removing the last  $n - t$  bits in each right node we obtain the graph  $G_{n,t}$ . We only need to check that in the bipartite graph  $G_{n,t} = (L_t = \{0, 1\}^n, R_t = \{0, 1\}^{m_t}, E_t \subseteq L_t \times R_t)$  (where  $m_t \leq n - \log \Delta$ ), every right node with non-zero degree has degree at least  $\Delta$ . This follows easily from the linearity of  $\text{EXT}_t(x, y)$  defined to be EXT( $x, y$ ) from which we cut the last  $n - t$  bits.

Indeed, let  $z$  in  $\{0, 1\}^{m_t}$  be a right node with non-zero degree. This means that there exist  $x$  and  $y$  such that  $\text{EXT}_t(x, y) = z$ . Since the function  $f_y(x) = \text{EXT}_t(x, y)$  is linear in  $x$ , it follows that  $\{x' \mid \text{EXT}(x', y) = z\} = \{x' \mid A_y \cdot x' = z\}$  (i.e., the preimage of  $z$ ) is an affine space over  $\text{GF}[2]$  with dimension at least  $n - m_t \geq \log \Delta$ , and therefore  $z$  has degree at least  $\Delta$ . Moreover, given  $y$ , we can find  $\Delta$  preimages of  $z$  in time polynomial in  $n$ , by solving the linear system. ◀

**Acknowledgments.** The author is grateful to Bruno Bauwens for his insightful observations.

---

## References

- 1 B. Bauwens, A. Makhlin, N. Vereshchagin, and M. Zimand. Short lists with short programs in short time. In *Proceedings of 28th IEEE Conference on Computational Complexity, Stanford, California, USA, 2013*.
- 2 Bruno Bauwens and Marius Zimand. Linear list-approximation for short programs (or the power of a few random bits). In *IEEE 29th Conference on Computational Complexity, CCC 2014, Vancouver, BC, Canada, June 11-13, 2014*, pages 241–247. IEEE, 2014. doi:10.1109/CCC.2014.32.
- 3 H. Buhrman, L. Fortnow, I. Newman, and N. Vereshchagin. Increasing Kolmogorov complexity. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science*, pages 412–421, Berlin, 2005. Springer-Verlag *Lecture Notes in Computer Science #3404*.
- 4 Gregory J. Chaitin. Information-theoretic characterizations of recursive infinite strings. *Theor. Comput. Sci.*, 2(1):45–48, 1976.
- 5 R. Downey and D. Hirschfeldt. *Algorithmic randomness and complexity*. Springer Verlag, 2010.
- 6 Ran Raz, Omer Reingold, and Salil P. Vadhan. Extracting all the randomness and reducing the error in Trevisan’s extractors. *J. Comput. Syst. Sci.*, 65(1):97–128, 2002. doi:10.1006/jcss.2002.1824.
- 7 Jason Teutsch. Short lists for shortest descriptions in short time. *Computational Complexity*, 23(4):565–583, 2014. doi:10.1007/s00037-014-0090-3.
- 8 Jason Teutsch and Marius Zimand. A brief on short descriptions. *SIGACT News*, 47(1):42–67, March 2016.
- 9 Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012. doi:10.1561/04000000010.
- 10 Marius Zimand. Short lists with short programs in short time – A short proof. In Arnold Beckmann, Erzsébet Csuhaj-Varjú, and Klaus Meer, editors, *Language, Life, Limits – 10th Conference on Computability in Europe, CiE 2014, Budapest, Hungary, June 23-27, 2014*.

## 58:12 List Approximation for Increasing Kolmogorov Complexity

*Proceedings*, volume 8493 of *Lecture Notes in Computer Science*, pages 403–408. Springer, 2014. doi:10.1007/978-3-319-08019-2\_42.

- 11 Marius Zimand. Kolmogorov complexity version of Slepian-Wolf coding. *CoRR*, abs/1511.03602, 2015. URL: <http://arxiv.org/abs/1511.03602>.